The bug in apache Xerces

Chris Perivolaropoulos

February 28, 2016

Contents

At the time of writing mwdumper a strange, semi-random bug. While `make sql-dump-parts` is running the following is encountered:

```
...

376,000 pages (14,460.426/sec), 376,000 revs
   (14,460.426/sec)
377,000 pages (14,458.848/sec), 377,000 revs
   (14,458.848/sec)
Exception in thread "main" java.lang.
   ArrayIndexOutOfBoundsException: 2048
        at org.apache.xerces.impl.io.UTF8Reader.read(
           Unknown Source)
        at org.apache.xerces.impl.XMLEntityScanner.
           load(Unknown Source)
        at org.apache.xerces.impl.XMLEntityScanner.
           scanContent(Unknown Source)
        at org.apache.xerces.impl.
           XMLDocumentFragmentScannerImpl.scanContent
           (Unknown Source)
        at org.apache.xerces.impl.
           XMLDocumentFragmentScannerImpl$FragmentContentDispatcher
           .dispatch(Unknown Source)
        at org.apache.xerces.impl.
           XMLDocumentFragmentScannerImpl.
           scanDocument(Unknown Source)
```

```
        at org.apache.xerces.parsers.
           XML11Configuration.parse(Unknown Source)
        at org.apache.xerces.parsers.
           XML11Configuration.parse(Unknown Source)
        at org.apache.xerces.parsers.XMLParser.parse(
           Unknown Source)
        at org.apache.xerces.parsers.
           AbstractSAXParser.parse(Unknown Source)
        at org.apache.xerces.jaxp.
           SAXParserImpl$JAXPSAXParser.parse(Unknown
           Source)
        at javax.xml.parsers.SAXParser.parse(
           SAXParser.java:392)
        at javax.xml.parsers.SAXParser.parse(
           SAXParser.java:195)
        at org.mediawiki.importer.XmlDumpReader.
           readDump(XmlDumpReader.java:88)
        at org.mediawiki.dumper.Dumper.main(Dumper.
           java:142)
make: *** [/scratch/cperivol/wikipedia-mirror/drafts/
   wikipedia-parts/enwiki-20131202-pages-articles20.
   xml-p011125004p013324998.sql] Error 1
```

Inspecting the makefules and running `make --just-print sql-dump-parts` we find out that the failing command is:

```
$ java -jar /scratch/cperivol/wikipedia-mirror/tools/
   mwdumper.jar   --format=sql:1.5 /scratch/cperivol/
   wikipedia-mirror/drafts/wikipedia-parts/enwiki
   -20131202-pages-articles20.xml-
   p011125004p013324998.fix.xml > /root/path/
   wikipedia-parts//enwiki-20131202-pages-articles20.
   xml-p011125004p013324998.sql
```

Fortunately this does not run for too long so we can safely experiment. Here is the `time` output:

```
26.65s user 1.73s system 78% cpu 35.949 total
```

The error seems to be during reading of the XML dump so it is not specific to SQL output. This could be useful for figuring out which article causes the error, removing which will hopefully resolve the error. To find that out we first try exporting to XML:

```
$ java -jar /scratch/cperivol/wikipedia-mirror/tools/
    mwdumper.jar   --format=xml /scratch/cperivol/
    wikipedia-mirror/drafts/wikipedia-parts/enwiki
    -20131202-pages-articles20.xml-
    p011125004p013324998.fix.xml > /tmp/just-a-copy.
    xml
```

As expected the same error as above is yielded. We then look for the last article two it tried to export by printing in reverse order the output xml file, finding the last two occurances of *<title>* with `grep` and reverse again to print them in the original order (note that `tac` is like `cat`, only that yields lines in reverse order):

```
$ tac /tmp/just-a-copy.xml | grep "<title>" -m 2 |
    tac
       <title>The roaring 20s</title>
       <title>Cranopsis bocourti</title> # <- This is
           the last one
```

This operation finishes quickly despite `/tmp/just-a-copy.xml` being fairly large because `tac` seeks to the end of the file and reads backwards until `grep` finds the 2 occurances it is looking for and quits. On ext3 the seek operation does not traverse the entire file. Indeed from the `tac` source code:

```
if (lseek (input_fd, file_pos, SEEK_SET) < 0)
     error (0, errno, _("%s: seek failed"), quotef (
         file));
/* Shift the pending record data right to make room
    for the new.
    The source and destination regions probably
       overlap.  */
memmove (G_buffer + read_size, G_buffer,
    saved_record_size);
past_end = G_buffer + read_size + saved_record_size;
/* For non-regexp searches, avoid unnecessary
    scanning. */
if (sentinel_length)
     match_start = G_buffer + read_size;
else
     match_start = past_end;

if (safe_read (input_fd, G_buffer, read_size) !=
    read_size)
```

```
{
    error (0, errno, _("%s: read error"), quotef (
       file));
    return false;
}
```

Let's save the path of the original xml file in a variable as we will be using it a lot. So from now on `$ORIGINAL_XML` will be the path of the original xml.

```
$ export ORIGINAL_XML=/scratch/cperivol/wikipedia-
   mirror/drafts/wikipedia-parts/enwiki-20131202-
   pages-articles20.xml-p011125004p013324998.fix.xml
```

First let's see if there is anything strange going on in the xml file:

```
$ grep "<title>Cranopsis bocourti</title>" -A 200 -B
   100 $ORIGINAL_XML | less
```

`| less` is to browse and `-A 200 -B 100` means *"show 200 lines after and 100 before the matching line"*. Nothing peculiar was found, so we can't really fix the problem in-place, we will try crudely removing the entire article and hope it works (spoiler alert: it does).

We will try to inspect the parents of the `title` of the breaking article. Fortunately the generated xml is indented so we can find the parents based on that. We count 6 spaces of indentation so we will search backwards from there on each level of indentation. The first line we find on each case will be a direct parent of the article.

```
$ for i in {0..6}; do \
    echo "Level $i:"; \
    tac /tmp/just-a-copy.xml | grep "^ \{$i\}<[^/]" -
       m 1 -n | tac; \
done

Level 0:
17564960:<mediawiki xmlns="http://www.mediawiki.org/
   xml/export-0.3/" xmlns:xsi="http://www.w3.org
   /2001/XMLSchema-instance" xsi:schemaLocation="http
   ://www.mediawiki.org/xml/export-0.3/ http://www.
   mediawiki.org/xml/export-0.3.xsd" version="0.3"
   xml:lang="en">
Level 1:
```

```
Level 2:
38:  <page>
Level 3:
Level 4:
35:    <revision>
Level 5:
Level 6:
26:      <text xml:space="preserve">&lt;!-- This
   article was auto-generated by [[User:Polbot]]. --&
   gt;
```

Looks like the xml is just `page` s thrown in a grand domain called `mediawiki`. We could have seen that from the java source too but as expensive as this is, it is much faster than dealing with the source of `mwdumper`.

The easiest way to cut off this article would be `awk` but that will take ages and we want to optimize and automate this entire process. First let's try just plain comparing the articles:

```
$ cmp /tmp/just-a-copy.xml $ORIGINAL_XML
/tmp/just-a-copy.xml /scratch/cperivol/wikipedia-
   mirror/drafts/wikipedia-parts/enwiki-20131202-
   pages-articles20.xml-p011125004p013324998.fix.xml
   differ: byte 2, line 1
```

That was fast... Let's see what went wrong:

```
$ head $ORIGINAL_XML
<mediawiki xmlns="http://www.mediawiki.org/xml/export
   -0.8/" xmlns:xsi="http://www.w3.org/2001/XMLSchema
   -instance" xsi:schemaLocation="http://www.
   mediawiki.org/xml/export-0.8/␣http://www.mediawiki
   .org/xml/export-0.8.xsd" version="0.8" xml:lang="
   en">
  <siteinfo>
    <sitename>Wikipedia</sitename>
    <base>http://en.wikipedia.org/wiki/Main_Page</
       base>
    <generator>MediaWiki 1.23wmf4</generator>
    <case>first-letter</case>
    <namespaces>
      <namespace key="-2" case="first-letter">Media</
         namespace>
      <namespace key="-1" case="first-letter">Special
         </namespace>
```

```
      <namespace key="0" case="first-letter" />

$ head /tmp/just-a-copy.xml
<?xml version="1.0" encoding="utf-8" ?>
<mediawiki xmlns="http://www.mediawiki.org/xml/export
   -0.3/" xmlns:xsi="http://www.w3.org/2001/XMLSchema
   -instance" xsi:schemaLocation="http://www.
   mediawiki.org/xml/export-0.3/ http://www.mediawiki
   .org/xml/export-0.3.xsd" version="0.3" xml:lang="
   en">
  <siteinfo>
    <sitename>Wikipedia</sitename>
    <base>http://en.wikipedia.org/wiki/Main_Page</
       base>
    <generator>MediaWiki 1.23wmf4</generator>
    <case>first-letter</case>
    <namespaces>
      <namespace key="-2">Media</namespace>
```

The attributes of the xml tags are quite different. Our best chance is if the line numbers match up. We count the numbers of lines in `/tmp/just-a-copy.xml` and hope that the corresponding line number in `$ORIGINAL_XML` will be the same line. If that is so we can ignore the the contextual xml information and just blank out the problematic article. We will use `wc` which is also quite fast.

```
$ wc -l /tmp/just-a-copy.xml
17564961 /tmp/just-a-copy.xml
```

And the corresponding line in `$ORIGINAL_XML` would be about:

```
$ sed "17564960q;d" $ORIGINAL_XML
[[Willie Jones (American football)|Willie Jones]],
```

Football... nothing to do with frogs. Looks like there is no avoiding some level of parsing.

0.1 Parsing

We will make the following assumptions to avoid properly parsing the document:

- The XML in the original file is valid

- Any XML within the articles is HTML escaped

First off working with lines is slow because user space code needs to look for newlines. Working bytes delegates work to the kernel, speeding things up considerably. So the `dd` is the right tool for the job. So we will first find at which byte is the article I am interested in.

```
$ grep -b "<title>Cranopsis␣bocourti</title>" -m 1
   $ORIGINAL_XML
1197420547:    <title>Cranopsis bocourti</title>
```

This may take a little while but you are stuck with it unfortunately. Our stategy is to make two files: `/tmp/original_tail.xml` that will contain all the data *after* the page we want to remove and `/tmp/original_head.xml` that will contain all the data *before* the page we want to remove.

Now we will use `sed` to look for `</page>` after byte 1197420547 which will be point x we will and dump the contents of `$ORIGINAL_XML` after point x:

```
$ dd if=$ORIGINAL_XML skip=1197420547 ibs=1 | sed
   '0,/<\/page>/d' > /tmp/original_tail.xml
```

Great, that worked! `dd` does not copy in reverse so we will need to do something more complex to construct `/tmo/original_head.xml`. Let's say the position where we found the title of the page we want to remove is $\alpha = 1197420547$ and the point where the page starts is point β. It is fairly safe to assume that $\beta > \alpha - 1000$ (we can calibrate the constant 1000 if that assumption is wrong, but it turns out that it isn't). This way we only need to search into 1Kb for `<page>`. Effectively instead of copying the bytes in range $[0, \beta)$ we are concatenating two ranges $[0, \alpha - 1000] \cup (\alpha - 1000, \beta)$ by making a subshell that will first output the first range and then output $(\alpha - 1000, \alpha)$ stopping when it finds `<page>`. Here is the one liner:

```
$ (dd count=$((1197420547-1000)) ibs=1 if=
   $ORIGINAL_XML; \
   dd if=$ORIGINAL_XML count=1000 skip=$
      ((1197420547-1000)) ibs=1 \
       | tac | sed '/<page>/,$d' | tac) > /tmp/
          original_head.xml
```

1 Covering up with spaces

From the above exploration of ways for circumventing the issue of the breaking article we omitted a fairly obvious, but thematically different approach: covering up breaking article with spaces. Once we find out the range in which the page resides we can `mmap` precisely in that part of `$ORIGINAL_XML` and then `memset` covering it up with space characters. The actual implementation lives in `data/page_remover.c`, below we present the call to mmap:

```
ctx->off = off-pa_off;
ctx->fd = open(fname, O_RDWR, 0x0666);
if (ctx->fd == -1) {
    perror("open");
    return NULL;
}

ctx->size = len;
ctx->data = mmap(0, len+ctx->off, PROT_READ |
   PROT_WRITE,
                MAP_SHARED, ctx->fd, pa_off);
if (ctx->data == MAP_FAILED) {
    perror ("mmap");
    return NULL;
}
```

and the `mmemset`:

```
/* You MIGHT want to thread this but I dont think it
   will make
 * much more difference than memset. */
memset(ctx->data + ctx->off, '␣', ctx->size);
```

Surprisingly this did not fix the mwdumper issue which points to a possible memory leak on the part of xerces but it is beyond the scope of this project to debug fix that if we have a choice.

2 The sed command

Above we kind of glazed over our use the `sed` command but it might be interesting to spend some ink on it. Sed is a unix tool found in coreutils that according to it's man page is a

stream editor for filtering and transforming text.

The basic premise is that the *"pattern space"*, or the input stream which is a normal unix stream coming from a file, a pipe or just stdin, is passed through a programmable pipeline. Either the modified pattern space itself is printed or, with the use of the `-n` flag, selected parts of it. Let's look at the use that we have made for sed above

Initially we used sed to print a specific line in a file:

```
$ sed "17564960q;d"
```

This sed program is separated by a semicolon. Sed iterates over the lines of the input stream and runs each of the `;` separated commands on them in sequence until one succeeds. The commands here are `17564960q` and `d`. `17564960q` will quit sed once line 17564960 is reached. `d` will discard the current line. So sed discards lines until it reaches line 17564960 which it prints and quits.

We then used a sed command as part of a series of shell commands piped together in order to print all the lines of a stream after a specific pattern (in our case `</page>`).

```
$ sed '0,/<\/page>/d'
```

This time we have only a single sed command, `d`. Sed iterates over the lines in the stream, discarding lines in the range of lines 0 to the line that matches `<\/page>`, effectively only printing lines after `</page>`.

Our final use of sed is the inverse of the aforementioned one,

```
$ sed '/<page>/,$d'
```

Here sed iterates again over all the lines of the stream this time discarding lines in the range between the first line that matches `<page>` until the final line, denoted with a `$`.