Wikipedia Mirror

Chris Perivolaropoulos

Sunday 21 February 2016

Contents

1 The xerces bug **1**
- 1.1 The final solution . . . 8
- 1.2 Covering up with spaces . . . 8
- 1.3 The sed command . . . 9

2 mediawiki stack overview **10**
- 2.1 Elements of the stack . . . 10
- 2.2 Tools . . . 12
- 2.3 Setting up . . . 15

3 Mediawiki Extensions **17**

4 Dumps **18**
- 4.1 PHP script . . . 18
- 4.2 mwdumper . . . 18

5 Automation **18**
- 5.1 Makefiles / laziness . . . 18
- 5.2 Shell scripts . . . 18
- 5.3 Bitnami . . . 18

6 Performance **18**
- 6.1 Compile time . . . 18
- 6.2 Runtime . . . 19

7 Appendix (script sources) **19**
- 7.1 $\text{page}_{\text{remover.c}}$ . . . 19
- 7.2 utf8thread.c . . . 22

7.3 sql-clear.sh 26
7.4 webmonitor.py 27
7.5 xml-parse.sh 32

1 The xerces bug

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.

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
```

1.1 The final solution

All the above was used to compose a script that lives in `data/xml-parse.sh` which is utilised by the makefiles to remove all problematic articles. If

mwdumper fails, we identify the article that caused the breakage and remove it using `xml-parse.sh`. Then we rerun mwdumper. We repeat that until mwdumper succeeds. In total the conflicting articles are aboit 10-15, and are different depending on the dump being used.

1.2 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.

1.3 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 `$`.

2 mediawiki stack overview

Wikipedia-mirror builds upon the mediawiki stack provided by bitnami. A service that builds the entire server within the confines of a directory. This is useful because we avoided the overhead of dealing with container or VM technologies and we had direct access to the filesystem of the stack while still having bitnami's build system do the tedious job of orchestrating the various components and separating our sever from the rest of the system.

The stack is comprised of

- An http server, in our case apache
- The web application runtime, in our case PHP
- A database, in our cas MySQL
- The web application itself, in our case mediawiki

All of the above are provided by the the bitnami mediawiki stack. Xampp used to be go-to for that but it is unmaintained so we decided to go with bitnami which works pretty well.

Once the stack is set up properly the wikipedia dump xml is downloaded and then turned into an sql dump with mwdumper. Could be piped directly to MySQL? but extracting can take time and things tend to go wrong during the dumping step.

2.1 Elements of the stack

We present each of the elements of the stack in more detail below.

(a) Apache
As per wikipedia:

```
The Apache HTTP Server, colloquially called Apache, is the world's
most used web server software. Originally based on the NCSA HTTPd
server, development of Apache began in early 1995 after work on the
NCSA code stalled. Apache played a key role in the initial growth of
the World Wide Web, quickly overtaking NCSA HTTPd as the dominant HTTP
server, and has remained most popular since April 1996. In 2009, it
became the first web server software to serve more than 100 million
```

```
websites.

Apache is developed and maintained by an open community of developers
under the auspices of the Apache Software Foundation. Most commonly
used on a Unix-like system (usually Linux), the software is available
for a wide variety of operating systems besides Unix, including
eComStation, Microsoft Windows, NetWare, OpenVMS, OS/2, and
TPF. Released under the Apache License, Apache is free and open-source
software.
```

it is fair to say that apache is at least one of the most popular web servers on the internet. wikipedia.org itself seems to be using a more complex stack involving varnish, an HTTP accelerator, and nginx, an alternative, also quite popular HTTP server. We arrive at this conclusion by inspecting the headers returned by wikipedia.org. In the `http://www.wikipedia.org` case we are redirected to the secure domain (pay attention to the `Server:` line):

```
$ curl -s -D - http://www.wikipedia.org -o /
   dev/null
HTTP/1.1 301 TLS Redirect
Server: Varnish
[...]
```

And if we directly ask for `https://www.wikipedia.org` nginx seems to be handling our request:

```
$ curl -s -D - https://www.wikipedia.org -o /
   dev/null
HTTP/1.1 200 OK
Server: nginx/1.9.4
[...]
```

However it is beyond the scope of the project to precisely replicate wikipedia's infrastructure. We focus on the functionality. Therefore due to the popularity, familiarity and by virtue of apace being part of the automatically installable bitnami mediawiki stack, we use it as our server.

(b) PHP

Mediawiki, which is discussed later, is written entirely in PHP, a popular server side, dynamically typed, object oriented scripting language. PHP is essential and is installed along the bitnami mediawiki stack. PHP is popular among web developers partly due to it's support for multiple relational database libraries (including PostgreSQL, MySQL, Microsoft SQL Server and SQLite) and it essentially being structred as a template language generating HTML.

(c) MySQL

Mediawiki can use a number of different SQL database backends:

- **MSSQL:** An SQL database by Microsoft
- **MySQL:** Using the standard PHP library for MySQL.
- **MySQLi:** An extension to the MySQL backend
- **Oracle:** A propertiary SQL database by Oracle.
- **SQLite:** An SQL database that is typically accessed as a library rather than over a client-server scheme as is the case with the other options on the list.

Wikipedia provides multiple dump files for SQL tables of secondary importance in MySQL format (eg. page redirects, categories etc) and suggests `mwdumper` which parses the XML dumpls of the wikipedia articles into MySQL. That and bitnami providing it as part of it's automatically built stack, make MySQL the obvious choice for the wikipedia-mirror stack.

(d) MediaWiki

Mediawiki is the beating heart of wikipedia.

2.2 Tools

A number of tools were developed in assisting the

(a) $\text{page}_{\text{remover.c}}$

As previously discussed, the `xerces` library that `mwdumper` depends on fails, seemingly at random, to process certain pages. To address this issue we remove the pages completely and retry. Since this task is fairly straight forward yet performance sensitive we resorted to writing a small low level program in C to address it, `page_remove.c`. Page remover accepts as input the path of the

XML wikipedia dump, the offset of the article and the size of the article. It then uses the `mmap` system call to random-access the data within the file and fill the article with withespace characters. `page_remover.c` is not threaded as the bottleneck is the HDD IO speed.

(b) sql-clear.sh

`sql-clear.sh` is a small bash script that truncates all tables from a database. Truncating means leaving the table scheamata unaffected and delete all internal data.

(c) utf8thread.c

`utf8thread.c` is another low level program that blanks out all invalid utf-8 characters from a file. We used `pthreads` to speed things up.

(d) webmonitor.py

`webmonitor.py` is a python script that sets up a web page that shows live data in the form of a histogram about the progress of the database population. `webmonitor.py` serves a static html page and then deeds it the data over websocket. Webmonitor can show any stream of `<epoc date> <float value>` pairs that it receives in it's input. As a sample:

```
$ pip install tornado
```

First install the dependencies of the script. That would be tornado, an asynchronous web framework supporting websockets. We will instruct `tornado` tornado will serve the following page:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD␣HTML␣
   4.01//EN" "http://www.w3.org/TR/html4/
   strict.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="
       text/html;␣charset=utf-8">
    <title>DrNinjaBatmans Websockets</title>

    <script type="text/javascript" src="http
       ://code.jquery.com/jquery-1.10.1.js"><
       /script>
```

```
    <script type="text/javascript" src="http
       ://code.highcharts.com/highcharts.js">
       </script>

    <script>
     var chart; // global
     var url = location.hostname + ':' + (
        parseInt(location.port));
     var ws = new WebSocket('ws://' + url +
        '/websocket');
     ws.onmessage = function(msg) {
         add_point(msg.data);
     };

     // ws.onclose = function() { alert('
        Connection closed.'); };

     var add_point = function(point) {
         var series = chart.series[0],
         shift = series.data.length > %d;

         chart.series[0].addPoint(eval(point)
            , true, shift);
     };

     $(document).ready(function() {
         chart = new Highcharts.Chart(JSON.
            parse('%s'));
     });
    </script>

  </head>
  <body>
      <div id="container" style="width:␣800px
         ;␣height:␣400px;␣margin:␣0␣auto"></
         div>
  </body>
</html>
```

In essence this page expects to read a stream of values from a websocket at `ws://localhost:8888/hostname` – although it is smart enough to change the `localhost:8888` if you are serving this to another location – and plot them in real time using `highcharts.js`.

The attentive reader may notice that the above is not quite HTML but rather a python formatted string. That is for two reasones. First because the script handles the configuration (see `chart = new Highcharts.Chart(JSON.parse('%s'));`). Second because the width of the graph will is calulated at page load time and the plot needs to be shifted to only show the most recent points.

```
$ for i in {1..100}; do echo $i;  sleep  1;
    done | \
     awk -oL "{print␣\$1/100}" | \
     python webmonitor.py
```

This will produce, in 1 second intervals, numbers from 1 to 100. Then it normalizes them using `awk` and feeds them to webmonitor. After this command executes we can open the browser and then navigate to `localhost:8888`.

We utilize this to remotely monitor the total size of data that `mysql` consumes.

(e) xml-parse.sh

2.3 Setting up

Following are step by step instructions First, clone the git repo:

```
$ git clone https://github.com/fakedrake/
   wikipedia-mirror
$ cd wikipedia-mirror
```

At this point in theory one can run `make sql-load-dumps` which will take care of stting up everything needed to load the the database dumps into the working SQL database. Of course for that to happen first a couple of steps need to be carried out:

- Download the wikipedia database dumps in XML format.
- Transform them into a format that MySQL understands.
- Set up the bitnami stack that includes a local install of MySQL
- Load the MySQL dumps into MySQL

All of these steps are encoded as part of the a dependency hierarchy encoded into makefile targets and are in theory taken care of automatically, effectively yielding a functioning wikipedia mirror. However this process is extremely long fragile so it is advised that each of these steps be run individually by hand.

First, download and install bitnami. The following command will fetch an executable from the bitnami website and make a local installation of the bitnami stack discussed above:

```
$ make bmw-install
```

Next step is to make sure `maven`, the java is a software project management and comprehension is installed, required to install and setup mwdumper (see below). You can do that by making sure the following succeeds:

```
$ mvn --version
```

Note: if running on Ubuntu 14.04, you may need to install Maven (for Java) using `sudo apt-get install maven`.

Now everything is installed to automatically download Wikipedia's XML dumps and then convert them to SQL using maven. First maven will be downloaded and built. Then the compressed XML dumps will be downloaded from the wikipedia, they will be uncompressed and finally converted to MySQL dumps using `mwdumper`. This is a fairly lengthy process taking 6 to 11 hours on a typical machine:

```
$ make sql-dump-parts
```

After that's done successfully you can load the SQL dumps to the MySQL database.

```
$ make sql-load-parts
```

Finally the

```
$ make mw-extensions
```

3 Mediawiki Extensions

For mediawiki to act like wikipedia a number of extensions are required. The installation process of such extensions is not automated or streamline. To automatically manage this complexity a mechanism is provided for declaratively installing extensions. To add support for an extension to wikipediabase one needs to add the following code in `Makefile.mwextnesions` (modifying accordingly):

```
MW_EXTENSIONS += newextension
mw-newextension-url = url/to/new/extnesion/
    package.tar.gz
mw-newextension-php = NewExtensionFile.php
mw-newextension-config = '$$phpConfigVariable = "
    value";'
```

And wikipedia-mirror will take care of checking if the extension is already installed and if not it will put the right files in the right place and edit the appropriate configuration files. The entry points for managing extensions are (provided that the name of the registered extension is newextension):

```
make mw-print-registered-extensions # Output a
    list of the registed extensions
make mw-newextension-enable         # Install and
    /or enable the extension
make mw-newextension-reinstall      # Reinstall
    an extension
make mw-newextension-disable        # Disable the
     extension
make mw-newextension-clean          # Remove the
    extension
```

All registered extensions will be installed and enabled when wikipedia-mirror is built.

4 Dumps

Wikipedia provides monthly dumps of all it's databases. The bulk of the dumps come in XML format and they need to be encoded into

MySQL to be loaded into the wikipedia-mirror database. There are more than one ways to do that.

4.1 PHP script

Mediawiki ships with a utility for importing the XML dumps. However it's use for importing a full blown wikipedia mirror is discouraged due to performance tradeoffs. Instead other tools like mwdumper are recommended that transform the XML dump into MySQL queries that populate the database.

4.2 mwdumper

The recomended tool for translating the XML dumps into MySQL code is mwdumper. Mwdumper is written in java and is shipped separately from mediawiki.

(a) Xml sanitizer
(b) Article dropper

5 Automation

5.1 Makefiles / laziness

5.2 Shell scripts

5.3 Bitnami

6 Performance

6.1 Compile time

Compile time includes the time it takes for:

- Downloading all the components of a wikipedia server
- The bitnami stack
  - mwdumper

- mediawiki-extensions
- Installing and building those components (~1 min)
- Downloading the wikipedia dumps
- Preprocessing the dumps (~10 mins)
- Populating the mysql database (~10 days)
  Builds were done on Infolab's Ashmore. The system's specs are quite high end but the bottleneck was the disk IO so less than 1% of the rest of the available resources were used during the MySQL database population.

(a) Attempts to optimizing MySQL

6.2 Runtime

Runtime of wikipedia mirror turned out to be too slow to be useful and therefore the project was eventually abandoned. Namely for the full wikipedia dump of July 2014 the load time for the Barack Obama, not taking advantage of caching was at the order of ~30s.

7 Appendix (script sources)

7.1 $\text{page}_{\text{remover.c}}$

```
/*
 * Copyright 2014 Chris Perivolaropoulos <
    cperivol@csail.mit.edu>
 *
 * This program is free software: you can
    redistribute it and/or
 * modify it under the terms of the GNU General
    Public License as
 * published by the Free Software Foundation,
    either version 3 of the
 * License, or (at your option) any later version
    .
 *
 * This program is distributed in the hope that
    it will be useful, but
 * WITHOUT ANY WARRANTY; without even the implied
     warranty of
```

```
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR
    PURPOSE.
 *
 * See the GNU General Public License for more
    details.  You should
 * have received a copy of the GNU General Public
     License along with
 * this program.
 *
 * If not, see <http://www.gnu.org/licenses/>.
 *
 * This should fill a range in a file with spaces
    . This is an in-place
 * operation so it should be pretty fast.
 *
 * Usage: page_remover PATH OFFSET LENGHT
 */

#include <assert.h>
#include <fcntl.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <semaphore.h>
#include <unistd.h>
#include <unistd.h>

#define USAGE_INFO "page_remover␣PATH␣OFFSET␣
   LENGTH"
#define PRINT(ctx, args...) do { sem_wait(&ctx->
   stdio_mutex);   \
                                printf(args);
                                          \
                                fflush(stdout);
                                          \
                                sem_post(&ctx->
                                   stdio_mutex);
                                        \
                                } while(0)
```

```
typedef struct context {
    int fd;
    size_t size;
    off_t off;
    sem_t stdio_mutex;
    void* data;
} context_t;

context_t* context_init(char* fname, off_t off,
   size_t len)
{
    context_t * ctx = (context_t*)malloc(sizeof(
       context_t));
    off_t pa_off = off & ~(sysconf(_SC_PAGE_SIZE)
        - 1);

    sem_init(&ctx->stdio_mutex, 0 /* Shared.
       Usually ignored */ , 1);

    PRINT(ctx, "Opening␣%s␣at␣%lu␣(len:␣%lu)\n",
       fname, off, len);

    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;
    }

    return ctx;
}

void context_destroy(context_t* ctx)
{
    if (close (ctx->fd) == -1)
      perror ("close");
```

```
    if (munmap ((void*)ctx->data, ctx->size) ==
        -1)
      perror ("munmap");

    sem_destroy(&ctx->stdio_mutex);
    free(ctx);
}

int main(int argc, char *argv[])
{
    if (argc != 4)
      fprintf(stderr, USAGE_INFO);

    context_t *ctx = context_init(argv[1], atoi(
        argv[2]), atoi(argv[3]));

    /* 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);

    context_destroy(ctx);
    return 0;
}
```

7.2 utf8thread.c

```
#include <assert.h>
#include <fcntl.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <semaphore.h>
#include <unistd.h>
#include <unistd.h>

sem_t stdio_mutex;
```

```
#define PRINT(args...) do {sem_wait(&stdio_mutex)
   ;      \
      printf(args);
                  \
      fflush(stdout);
                  \
      sem_post(&stdio_mutex);
                  \
    } while(0)

/* #define DEBUG(args...)                PRINT(
   args) */
#define DEBUG(...)

#define DEFAULT_CHAR '␣'
#define WORKERS 8
#define MESSAGE_DENSITY 1000000000

typedef unsigned long long u64;

#define UTF_LC(l) ((0xff >> (8 - (l))) << (8 - (l
   )))
#define UTF_CHECK(l, c) (((UTF_LC(l) & (c)) ==
   UTF_LC(l)) && (0 == ((c) & (1 << (7-(l))))))


#define UTF_LEN(x) (UTF_CHECK(6, x) ? 6 :       \
                  UTF_CHECK(5, x) ? 5 : \
                  UTF_CHECK(4, x) ? 4 : \
                  UTF_CHECK(3, x) ? 3 : \
                  UTF_CHECK(2, x) ? 2 : -1)

struct crange {
    u64 start, end;
};

/* Get return the next character after the last
   correct one. */
inline u64 valid_utf8(u64 c)
{
    char i;
    /* Ascii */
    if ((*(char*)c & 0x80) == 0)
      return c+1;
```

```
    /*  */
    for (i = UTF_LEN(*(char*)c)-1; i>0; i--) {
      c++;
      if (!UTF_CHECK(1, *(char*)c)) {
          return (u64)NULL;
      }
    }

    return i<0 ? 0 : c+1;
}


void* fix_range(void* _r)
{
    struct crange* r = _r;
    u64 tmp, id = r->start;
    long long unsigned count = 0;

    while ((u64)r->start < (u64)r->end) {
      if (count++ % MESSAGE_DENSITY == 0)
          printf ("[worker:␣0x%016llx]␣Done␣with␣
              %lluK.\n", id, count % 1024);

      if (!(tmp = valid_utf8(r->start))){
          PRINT("Invalid␣char␣0x%x␣(next:␣0x%x)\n
              ",
                 *(char*)r->start, *(char*)(r->
                    start+1));
          *((char*)r->start) = DEFAULT_CHAR;
          (r->start)++;
      } else {
          r->start = tmp;
      }
    }

    PRINT ("[worker:␣0x%016llx]␣OUT\n", id);
    return NULL;
}

void run(u64 p, u64 sz)
{
    int n, i;
    u64 wsize;
    pthread_t workers[WORKERS];
```

```
    struct crange rngs[WORKERS];

    wsize = sz/WORKERS + 1;
    printf("Base␣address:␣0x%016llx,␣step␣size:␣0
       x%016llx\n", p, wsize);

    for (i=0; i<WORKERS; i++){
      rngs[i].start = p + wsize*i;
      rngs[i].end = p + wsize*i + wsize;

      PRINT("Spawning␣worker␣%d␣on␣range␣[0x%016
         llx,␣0x%016llx),␣%llu␣bytes...", i, rngs
         [i].start, rngs[i].end, wsize);
      if ((n = pthread_create(workers+i, NULL,
         fix_range, (void*)(rngs+i)))) {
          PRINT("FAIL\n");
          perror("worker");
          return;
      }
      PRINT("OK\n");
    }

    PRINT ("Wrapping␣up...\n");
    for (i=0; i<WORKERS; i++) {
      PRINT ("Joining␣worker␣%d...", i);
      pthread_join(workers[i], NULL);
      PRINT ("OK\n");
      PRINT("Worker␣%d␣went␣through␣%llu␣bytes.\n
         ",
            i, (u64)rngs[i].end - (u64)rngs[i].
               start);
    }
}


int main(int argc, char *argv[])
{
    int fd;
    long long int sz, p;
    struct stat buf;

    sem_init(&stdio_mutex, 0 /* Shared. Usually
       ignored */ , 1);

    fd = open(argv[1], O_RDWR, 0x0666);
```

```
    if (fd == -1) {
      perror("open");
      return 1;
    }

    fstat(fd, &buf);
    sz = buf.st_size;
    printf("File␣size:␣0x%016llx\n", sz);

    p = (u64)mmap (0, buf.st_size, PROT_READ |
        PROT_WRITE , MAP_SHARED, fd, 0);
    if (p == -1) {
      perror ("mmap");
      return 1;
    }

    run(p, buf.st_size);

    if (close (fd) == -1) {
      perror ("close");
      return 1;
    }


    if (munmap ((void*)p, buf.st_size) == -1) {
      perror ("munmap");
      return 1;
    }

    sem_destroy(&stdio_mutex);

    return 0;
}
```

7.3 sql-clear.sh

```
#!/bin/bash
MUSER="$1"
MPASS="$2"
MDB="$3"
MYSQL=$4
```

```
# Detect paths
AWK=$(which awk)
GREP=$(which grep)

if [ $# -ne 4 ]
then
      echo "Usage: $0 {MySQL-User-Name} {MySQL-
         User-Password} {MySQL-Database-Name} {
         MySQL executable to use}"
      echo "Drops all tables from a MySQL"
      exit 1
fi

TABLES=$($MYSQL -u $MUSER -p$MPASS $MDB -e 'show
   tables' | $AWK '{ print $1}' | $GREP -v '^
   Tables' )

for t in $TABLES
do
      echo "Clearing $t table from $MDB database
         ..."
      $MYSQL -u $MUSER -p$MPASS $MDB -e "truncate
         table $t"
done
```

7.4 webmonitor.py

```
"""
Just feed pairs of

<epoc date> <float value>

or even just

<float value>

One way to do that would be

    $ <cmd> stdbuf -oL awk "{print \$1/$$max}" |
    python webmonitor.py
```

```
and␣I␣will␣plot␣them␣on␣port␣8888.␣This␣will␣also
    ␣pipe␣the␣input␣right
out␣to␣the␣output.␣Strange␣input␣will␣be␣ignored␣
    and␣piped␣this␣way,
but␣this␣needs␣to␣be␣done␣by␣awk␣aswell␣in␣the␣
    above␣example.
"""

import sys
import json
import time

from threading import Thread
from collections import deque

import tornado.websocket as websocket
import tornado.ioloop
import tornado.web

HTML = """
<!DOCTYPE␣HTML␣PUBLIC␣"-//W3C//DTD␣HTML␣4.01//EN"
    ␣"http://www.w3.org/TR/html4/strict.dtd">
<html>
␣␣<head>
␣␣␣␣<meta␣http-equiv="Content-Type"␣content="text
    /html;␣charset=utf-8">
␣␣␣␣<title>DrNinjaBatmans␣Websockets</title>

␣␣␣␣<script␣type="text/javascript"␣src="http://
    code.jquery.com/jquery-1.10.1.js"></script>
␣␣␣␣<script␣type="text/javascript"␣src="http://
    code.highcharts.com/highcharts.js"></script>

␣␣␣␣<script>
var␣chart;␣//␣global
var␣url␣=␣location.hostname␣+␣':'␣+␣(parseInt(
    location.port));
var␣ws␣=␣new␣WebSocket('ws://'␣+␣url␣+␣'/
    websocket');
ws.onmessage␣=␣function(msg)␣{
␣␣␣␣add_point(msg.data);
};

//␣ws.onclose␣=␣function()␣{␣alert('Connection␣
    closed.');␣};
```

```
var␣add_point␣=␣function(point)␣{
␣␣␣␣var␣series␣=␣chart.series[0],
␣␣␣␣␣␣␣␣shift␣=␣series.data.length␣>␣%d;
␣␣␣␣chart.series[0].addPoint(eval(point),␣true,␣
    shift);
};

$(document).ready(function()␣{
␣␣␣␣chart␣=␣new␣Highcharts.Chart(JSON.parse('%s')
    );
});
␣␣␣␣</script>

␣␣</head><body><div␣id="container"␣style="width:␣
    800px;␣height:␣400px;␣margin:␣0␣auto"></div></
    body></html>
"""

config = {
    'visible_points': 10,
    'py_chart_opts': { 'chart': { 'renderTo': '
       container',
                                  '
                                     defaultSeriesType
                                     ': 'spline'
                                     },
                       'title': { 'text': '
                          DrNinjaBatmans␣data'},
                       'xAxis': { 'type': '
                          datetime',
                                  '
                                     tickPixelInterval
                                     ': '150'},
                       'yAxis': { 'minPadding':
                          0.2,
                                  'maxPadding':
                                     0.2,
                                  'title': {'text
                                     ': 'Value',
                                           '
                                              margin
                                              ':

                                              80}
```

```
                                },
                            'series': [{ 'name': 'Data
                                ',
                                        'data': []}]}

}

def date_float(s):
    try:
        date, val = s.split()
    except ValueError:
        val = s.strip()
        date = time.time()

    return int(date), float(val)


def send_stdin(fn=date_float):
    for raw in sys.stdin:
        sys.stdout.write(raw)

        # Ignore strange input.
        try:
            jsn = json.dumps(fn(raw))

            buf.append(jsn)

            for w in websockets:
                try:
                    w.write_message(jsn)
                except websocket.
                   WebSocketClosedError:
                    pass

        except:
            pass

    for ws in websockets:
        ws.close()

class StdinSocket(websocket.WebSocketHandler):
    def open(self):
        for i in buf:
            self.write_message(i)
```

```
        websockets.append(self)


    def closs(self):
        websockets.remove(self)

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write(HTML % (int(config['
            visible_points']),
                           json.dumps(config['
                               py_chart_opts'])))


if __name__ == "__main__":
    application = tornado.web.Application([
        (r"/", MainHandler),
        (r'/websocket', StdinSocket),
    ])
    buf = deque(maxlen=int(config['visible_points
        ']))
    websockets = []


    config['args'] = []
    for a in sys.argv[1:]:
        if '=' in a:
            k, v = a.split('=', 1)
            config[k] = v
        else:
            config['args'].append(a)

    Thread(target=send_stdin).start()
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

7.5 xml-parse.sh

```
  #!/bin/bash
#
```

```
# Simply removing specific articles fixes the
    xerces error with
# UTF8. If the articles are alone the error goes
    away
# aswell. Extremely weird but that's life.
    Fortunately the article is
# just a stub about some toad (Cranopsis bocourti
    )
#
# xml-parse.sh ORIGINAL_XML
    TITLE_OF_ARTICLE_TO_REMOVE [inplace]
#
# if 'inplace' is there the c program will be
    used to cover the article
# with spaces. This is much faster. Should be
    anyway. Otherwise the
# page is just ommited and the result is dumped
    in stdout. Helping
# messages are dumped in stderr After this you
    can run:
#
# java -jar tools/mwdumper.jar RESULTING_XML --
    format=sql:1.5 > SQL_DUMP

set -e
set -o pipefail

if [[ $# -lt 2 ]]; then
    echo "xml-parse.sh␣ORIGINAL_XML␣
        TITLE_OF_ARTICLE_TO_REMOVE␣[inplace]" 1>&2
    exit 0
fi

function my_dd {
    coreutils_version=$(dd --version | head -1 |
        cut -d\  -f3 | colrm 2 2 )
    if [[ $coreutils_version -ge 822 ]]; then
      eval "dd␣iflag=count_bytes␣iflag=direct␣
          oflag=seek_bytes␣ibs=1M␣$@"
    else
      echo "Your␣coreutils␣may␣be␣a␣bit␣old␣(
          $coreutils_version).␣822␣is␣the␣one␣cool
          ␣kids␣use." >&2
      eval "dd␣$@␣ibs=1"
    fi
```

```
}

ORIGINAL_XML=$1

# Dump a part of the file in sdout using dd.
# Usage:
# file_range <filename> <first_byte> <start|end|
    length>
#
# Length can be negative
function file_range {
    file=$1
    start=$2
    len=$3


    case $len in
      "end") my_dd if=$file skip=$start || exit
          1; return 0;;
      "start") my_dd if=$file count=$start ||
          exit 1; return 0;;
      "") echo "len␣was␣empty␣(file:␣$file,␣start
          :␣$start,␣len␣$len).␣Correct␣format␣<
          filename>␣<byte␣start>␣<length|'start'|'
          end'>" 1>&2; exit 1;;
      *) ;;
    esac

    if [[ $len -gt 0 ]]; then
      # Dump to stdout
      my_dd if=$file skip=$start count=$len ||
          exit 1
    else
      skip=$(($start + ($len)))
      len=$((- ($len)))

      if [[ $skip -lt 0 ]]; then
          skip=0
          len=$start
      fi

      # Dump to stdout
        my_dd if=$file skip=$skip count=$len ||
            exit 1
    fi
```

```
}

function backwards {
    tac -b | rev
}

function byte_offset {
    grep -b -o -m 1 -F  "$1" | cut -d : -f1
}

# Throw everything but the page in stdout
#
# neg_xml_page "Barack Obama"
function neg_xml_page {
    term="<title>$1</title>"
    title_offset=$(cat $ORIGINAL_XML |
       byte_offset "$term")
    echo -e "\n\tMethod:␣$2(blank␣is␣ok)" 1>&2
    echo -e "\tsearch␣term:␣$term" 1>&2
    echo -e "\tfile:␣$ORIGINAL_XML" 1>&2
    echo -e "\ttitle␣offset:␣$title_offset" 1>&2

    # Fail the term is invalid
    if [ -z "$title_offset" ]; then
      echo "Found␣'$title_offset'␣Grep-ing␣(cat␣␣
         $ORIGINAL_XML␣|␣grep␣-b␣-m␣1␣-F␣\"$term\
         "␣|␣cut␣-d:␣-f1)" 1>&2
      exit 1
    fi

    to_page_start=$(($(file_range $ORIGINAL_XML
       $title_offset -1000 | backwards |
       byte_offset "$(echo␣'<page>'␣|␣rev)")+7))
    echo -e "\tto␣page␣start␣(relative):␣
       $to_page_start" 1>&2

    file_range $ORIGINAL_XML $title_offset end |
       byte_offset "</page>" >&2
    echo $(($(file_range $ORIGINAL_XML
       $title_offset end | byte_offset "</page>")
       +7)) >&2
    to_page_end=$(($(file_range $ORIGINAL_XML
       $title_offset end | byte_offset "</page>")
       +7)) # len('</page>') == 7
```

```
    echo -e "\tto␣page␣end␣(relative):␣
       $to_page_end" 1>&2

    page_start=$(($title_offset - $to_page_start
       +1 ))
    echo -e "\tpage␣start:␣$page_start" 1>&2

    page_end=$(($title_offset + $to_page_end))
    echo -e "\tpage␣end:␣$page_end" 1>&2

    echo -e "\tbytes␣to␣copy:␣$(($(du␣-b␣
       $ORIGINAL_XML␣|␣cut␣-f1)␣-␣$page_start␣+␣
       $page_end))" 1>&2

    echo "Going␣to␣copy␣$page_start␣bytes" 1>&2
    file_range $ORIGINAL_XML $page_start start
    echo "Finished␣the␣first␣half␣up␣to␣
       $page_start,␣$((␣$(du␣-b␣$ORIGINAL_XML␣|␣
       cut␣-f␣1)␣-␣$page_end␣))␣to␣go" 1>&2
    file_range $ORIGINAL_XML $page_end end
    echo "Finished␣the␣whole␣thing." 1>&2
}

# Put stdin betwinn mediawiki tags and into
   stdout
function mediawiki_xml {
    (head -1 $ORIGINAL_XML; sed -n "/<siteinfo
       >/,/<\/siteinfo>/p;/<\/siteinfo>/q"
       $ORIGINAL_XML ; cat - ; tail -1
       $ORIGINAL_XML )
}

# 1: XML File
# 2: Article
# 3: Method (leave blank)
# Assert that the file is there and is not empty
fsize=$(du -b $ORIGINAL_XML | cut -f1)
if [[ 0 -eq $fsize ]]; then
    echo "ERROR:␣empty␣xml␣file␣$ORIGINAL_XML"
       1>&2
    exit 1
fi

echo "Will␣remove␣article␣'$2'␣from␣file␣$1␣(size
   :␣$fsize)" 1>&2
```

```
if ! neg_xml_page "$2" "$3"; then
    ret=$?
    echo "XML␣parsing␣script␣failed" 1>&2
    exit $ret;
fi
```