JB on programming Documentation

Release 1.0.0

Jacek Bzdak <jacekfoo@gmail.com>

Dec 29, 2018

Contents

Contents:

CHAPTER 1

Attention: This blog has moved

I have moved this blog to a new home.

Most of valuable content was moved there, so bookmark new site.

CHAPTER 2

How to use pythonbrew and virtualenv

Warning: This is outdated and not longer revelant.

In most of my projects I use virtualenvs and install dependencies by pip, well it's nothing special, since virtualenvs have become mostly standard in python developement.

But since among my colleagues I'm sort of early-adopter, and I find myself explaining this again and again, I figured that I'll write this article.

2.1 What is a python virtual environment

Virtual environment is *a way to manage python interpreter and associated installed packages*. Generaly you use separate virtualenvs for all your projects, in which case all these projects can depend on different versions of libraries.

2.2 Advantages of using virtualenv

- Projects using different virtualenvs can use different versions of libraries.
- To install a library inside virtualenv you don't need to have root privelages, which is good even if you are root, because packages installed via *pip* shouldn't be trusted.

2.3 Installing virtualenv

Normally you install virtualenv using your system package manager.

2.4 Using virtualenv

```
virtualenv foo # create virtualenv in foo subfolder of local folder
New python executable in foo/bin/python
Installing distribute (..) done.
Installing pip...............done.

source foo/bin/activate # Activate the virtualenv

pip install django #install packages
Downloading/unpacking django
  Running setup.py egg_info for package django

    warning: no previously-included files matching '__pycache__' found under␣
↪directory '*'
    warning: no previously-included files matching '*.py[co]' found under directory '*
↪'
Installing collected packages: django
  Running setup.py install for django
    changing mode of build/scripts-2.7/django-admin.py from 644 to 755

    warning: no previously-included files matching '__pycache__' found under␣
↪directory '*'
    warning: no previously-included files matching '*.py[co]' found under directory '*
↪'
    changing mode of /tmp/foo/bin/django-admin.py to 755
Successfully installed django
Cleaning up...

python -c 'import django' #Verify django is installed

deactivate # turn off the virtualenv

python -c 'import django' # See that django was installed locally only!
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ImportError: No module named django
```

Used commands

virtualenv path creates virtualenv in specified *path*. Interesting options include:

- *virtualenv –python=python2.5* creates virtualenv using specified interpreter
- *virtuaelnv –system-site-packages* allows virtualenv to use packages installed systemwide (it is nice since some packages can't be installed easily via pip).

source path/bin/activate Enables virtualenv in current console

pip install packagename installs package inside virtualenv (if it was activated)

deactivate disables virtualenv in current console

2.5 Pip cache trick

When using virtualenvs you'll find that installing packages via pip can be painfully slow. You can instruct pip to use cache for downloaded packages by adding following line to your *.bashrc* file.

```
export PIP_DOWNLOAD_CACHE=$HOME/.pipcache
```

2.6 Using pythonbrew

Virtual enviorment *borrows* one of your system interpreters, pythonbrew takes it step further: it downloads and compiles your own python interpreter (and has integrated support for virtualenvs.

2.7 Installing pythonbrew

Dont use instructions from pythonbrew repository, as they basically tell you: 'download file via http and execute it in bash' (which is *very insecure*).

Just clone repository stored in github at https://github.com/utahta/pythonbrew and execute *install_pythonbrew.py* script.

Then as instructed paste following into bashrc file:

```
[[ -s "$HOME/.pythonbrew/etc/bashrc" ]] && source "$HOME/.pythonbrew/etc/bashrc"
```

2.8 Using pythonbrew

pythonbrew install 2.7.1 Installs python *2.7.1* into pythonbrew

pythonbrew use 2.7.1 Use installed python *2.7.1* in current console.

pythonbrew venv create name -p 2.7.1 Create virtualenv named *name* for python interpreter version *2.7.1*

pythonbrew venv use name -p 2.7.1 Activate virtualenv named *name* for python interpreter version *2.7.1*

CHAPTER 3

How to use Libre Office serial document (mail merge) functionality

Libre Officess mailmerge functionality is a nice feature that allows you to create *any kind of serial document*, while all documentation seems to indicate that it is only usable for creating e-mails.

I used this tutorial (it has nice screenshots and so on).

So anyways steps are as follows:

3.1 Create data source

Create a data source, this can be:

1. A Libre Office Base database (or any database)
2. A csv file (**remember to have header row**).
3. An Libre Office Spreadsheet file (**also add a header row**)

3.2 Create a document template

Write the text

3.3 Attach database to the document

In writer Edit -> Exchange Database -> Browse -> Select your database file.

In writer View -> Datasources, then select your database file again.

You should see table contents.

3.4 Add fields to the document

Now you should be able to Drag and Drop fields (columns) from the table to the document.

3.5 Create serialized documents

Mail Merge Wizard or click the envelope icon in data sources. This stuff is mostly stupid and deals with preformatted address blocks, etc. I ve never used it.

1. Select starting Document: Select current document.
2. Select document type: Select letter
3. Insert address block: Uncheck all checkboxes.
4. Create salutation: Uncheck all checkboxes.

CHAPTER 4

When your cuda installation stops working (on DEBIAN)

There can be many signs for this error:

`deviceQuery` returns:

```
cudaGetDeviceCount returned 38-> no CUDA-capable device is detected
```

or you get:

```
no CUDA-capable device is detected
```

In my case it was version mismatch between installed `nvidia` kernel module and `libcuda1` library.

Note: Normally this module is installed via `DKMS` so kernel module version should match version of `nvidia-kernel-dkms`, but this is not always the case...

To check version of installed kernell module:

```
sudo modinfo nvidia-current | grep version
```

For now it should be `319.xxx`. If it has version mismatch `libcuda1` you have source of your errors (yay!).

CHAPTER 5

Set dirs to 755 and files to 644

So I dont forget, if you want to share a directory structure with people you can change permissions in following way:

```
find dir -type d -exec chmod 755 {} \;
find dir -type f -exec chmod 644 {} \;
```

In this way others will be able to enter any directory and won't be able to execuye files.

CHAPTER 6

Autocropping all images in folder

Another piece of code I'm putting here for posteriority. Anyways, let's say I have couple hundred imahes in folder (autogenerated) and I need to crop them automatically (remove extra background).

Here is a quic command that does the trick:

```
mogrify -trim +repage *.png
```

This comes with imagemagic suite.

CHAPTER 7

How to deploy django application on debian server (UWSGI version)

7.1 Install proper version of python

On debian I'd discourage using `system` python for deployment — mostly becaluse they tend to upgrade minor python versions without notice, which breaks `C ABI` in installed virtualenvs.

So either roll your own `deb` files that install pythons somewhere in `/usr/local` or compile python on server (if you frown on having developement tools on your server roll `debs`).

Pythonz is a nice tool to compile (and manage) many versions of python.

7.2 Install proper version of virtualenv

Note: This is more-or less irrevelant as Python (from version 3.4 onwards) has it's own built-in virtualenv tool, so if you use non-system one it'll have its own virtualenv.

Debian comes with ancient python/virtualenv version, and (at least on `testing`) it often breaks this install. If you install virtualenv locally it'll be much more stable.

To install virtualenv locally just download virtualenv, unpack package and use `virtualenv.py`.

This hsa the added benefit that you can have two versions of virtualenv one for python2 and one for python3.

7.3 Install your application into virtualenv

You know how to do that don't you?

Now you can tests whether your setup is correct, just run

```
./manage.py runserver
```

and see if you can connect to your application.

7.4 Install uwsgi into your virtualenv

Install `uwsgi` **into your virtualenv** from `pip`. Now you can run your application using uwsgi:

```
uwsgi --http :8000 --module webapp.wsgi
```

I strongly discourage you from using uwsgi bundled with system.

7.5 Use supervisord to launch your applicaiton

You'll need to run your application on system start, simplest way is to use `supervisord` (you can install it via aptitude).

First create a script that will:

1. Source virtualenv
2. cd to your app directory
3. run uwsgi.

Something along the lines:

```
#!/bin/bash
export HOME=/home/user
source $HOME/venv/bin/activate
cd $HOME/webapp
uwsgi --socket :8000 --module webapp.wsgi
exit $?
```

Notice that `--http` turned into `--socket`. Now uwsgi will speak `uwsgi` protocol (which should be faster than http).

Then add configuration to supervisord. All services are defined as `*conf` files (in `ini` format) inside `/etc/supervisor/conf.d`.

Create file containing something like:

```
[program:webapp]
command=/home/webapp/webapp.sh
autostart=true
autorestart=true
stderr_logfile=/var/log/webapp.err.log
stdout_logfile=/var/log/webapp.out.log
user=webapp
```

For all configuration options see the documentation.

Now after calling: `service supervisor restart` your django application should be running.

7.6 Connect uwsgi and nginx

Note: This part is more or less ripoff from: http://uwsgi-docs.readthedocs.org/en/latest/tutorials/Django_and_nginx.html

Add following sections to nginx configuration:

```
upstream django {
    # server unix:///path/to/your/mysite/mysite.sock; # for a file socket
    server 127.0.0.1:8000; # for a web port socket (we'll use this first)
}

# configuration of the server
server {
    # the port your site will be served on
    listen      80;
    # the domain name it will serve for
    server_name .example.com; # substitute your machine's IP address or FQDN
    charset     utf-8;

    # max upload size
    client_max_body_size 75M;   # adjust to taste


    # Finally, send all non-media requests to the Django server.
    location / {
        uwsgi_pass  django;
        include     /path/to/your/mysite/uwsgi_params; # the uwsgi_params file you␣
↪installed
    }
}
```

Now you should see something on your server port `80`. To finalize our setup we need to create static and media directories.

7.7 Connect ngingx and uwsgi via linux file sockets

Because of many (peformance, safety) reasons it is better to connect nginx with uwsgi via linux domain sockets.

First replace `uwsgi` call with something like that:

```
uwsgi --module webapp.wsgi --socket $HOME/sock/webapp.sock  --chown-socket=webapp-
↪user:www-data --chmod-socket=660
```

This does the following: creates a socket in `$HOME/sock/webapp.sock`, sets its group ownership to `www-data` (which is user/group used by both `nginx` and `apache` on default debian configuration).

Note: Linux file sockets use normal file permissions, so `nginx` has to have read-write access to it.

Then replace:

```
upstream django {
    server 127.0.0.1:8001; # for a web port socket (we'll use this first)
}
```

with:

```
upstream django {
    server unix:///path/to/your/mysite/webapp.sock; # for a file socket
}
```

7.8 Update media and static root

7.8.1 Update settings py

You'll need to update `STATIC_ROOT`, `STATIC_URL`, `MEDIA_ROOT`, `MEDIA_URL` settings of youe app.

Something along the lines:

```
MEDIA_ROOT = '/var/drigan-media'
MEDIA_URL = '/media/'
STATIC_ROOT = '/var/drigan-static'
STATIC_URL = '/static/'
```

7.8.2 Update nginx

```
location /media  {
    alias /path/to/your/mysite/media;  # your Django project's media files - amend as␣
↪required
}

location /static {
    alias /path/to/your/mysite/static; # your Django project's static files - amend␣
↪as required
}
```

7.9 Tweak uwsgi so it scales

You might want tweak `uwsgi` so it launches more processes/workers, but this well outside scope of this tutorial.

CHAPTER 8

Compress a dir to tar.xz

Everyone is using tar.gz and tar.bz2 formats, and these compression algorithms (while stable and installed everywhere) are definetely not state-of-the-art.

Anyways most modern systems have much better compression ratios (or much less compression/decompression overhead). Namely `xz` (better ration) `lzo` (much better resource usage).

Anyways since I allways forget syntax to do compression here is proper command, but today I found this nice tar switch `-a` which means: "Try to guess compression format from file extension", so:

```
tar -caf foo.tar.xz data
```

will compress to `xz`, while

```
tar -caf foo.tar.lzo data
```

will compress to `lzo`. At least `tar` has sane API.

This is nice, but sometimes you'll want to tweak compression ratio for used compressor — in this case just use pipes. If you pass `-` (if anything else is non-obvious just use `man`).

```
tar -c data - | xz -9c > data2.tar.xz
```

CHAPTER 9

Connecting remote display (projector) to `i3`

It turns out to be suprisingly easy `randr` and `i3` are integrated well enough.

I have decided that I'll add a workspace dedicated for projector.

`i3.config` snippet:

```
# switch to workspace
bindsym $mod+p workspace 11

bindsym $mod+Shift+p move container to workspace 11

workspace 11 output VGA-1
```

To enable display issue:

```
xrandr --output VGA-1 --auto --right-of LVDS-1
```

To disable:

```
xrandr --output VGA-1 --off
```

CHAPTER 10

CGI on Nginx or how to run man2html on debian

I don't enjoy reading manpages, `man` has old clunky interface, however do enjoy reading documentation in html, some good people developen `man2html` tool that is a `CGI` script that serves man pages in a browser.

This is why in 2015 year I spend part of an evening trying to configure modern web server to run CGI scripts. This might not be a suprise, that nginx doesn't serve CGI by default. It can serve `FastCGI`, which is a streamlied version of CGI, that unfortunately is incompatible with plain old CGI scripts.

However there were some other good people that wrote `fcgiwrap`, which is a a wrapper that talks FCGI protocol, and then launches plain old `cgi` scripts.

So to launch `man2html` on Debian, you'll need to:

1. Install required software: `aptitude install man2html nginx-full fcgiwrap`
2. There is a very good example fcgi config at `/usr/share/doc/fcgiwrap/examples/nginx.conf`, so just copy it to: `/etc/nginx/fcgiwrap.conf`
3. JUst include `/etc/nginx/fcgiwrap.conf` in your server config.

CHAPTER 11

New Java Features

I decided to refresh my Java knowledge (last version I used was java 1.6), and because I learn by coding (much better than I learn by reading) here are code samples I prepared.

Note: This might be updated.

11.1 Java 1.7

Note: All examples were actually made on Java `1.8` JVM.

I used this as a reference list of changes (Oracle comparison is way to comprehensive).

11.1.1 `java.nio.path` package

Java `File` class is awful, but a very nice api was introduced in this version of Java.

Package `java.nio.path` has a very nice tutorial by Oracle, so I won't describe it here.

Here is my (very simple) example, it works similarly to Disk Usage Analyzer. or `du` command on Linux, that is: it summarizes disk usage for a folder.

To do this I just needed to implement a `FileVisitor` instance, and then pass it to `Files.walkFileTree`.

Code Highlights

Most of the logic is in `Visitor` that subclasses `FileVisitor`, this class is used to traverse whole directory tree. Inside this instance we keep track of where in the directory tree we are, by using a stack.

```
Queue<Long> fileSizes = Collections.asLifoQueue(new ArrayDeque<>());
```

Each entry in the stack corresponds to a parent directory of currently processed path, and each contains a total size of that directory.

To add size of current object to size of the parent following code is used:

```
private void pushSize(long size){
   long lastSize = fileSizes.poll();
   lastSize+=size;
   fileSizes.add(lastSize);
}
```

```
@Override
public FileVisitResult preVisitDirectory(Path dir, BasicFileAttributes attrs) throws␣
↪IOException {
    fileSizes.add(0L);
    return FileVisitResult.CONTINUE;
}

@Override
public FileVisitResult postVisitDirectory(Path dir, IOException exc) throws␣
↪IOException {
    long dirSize = fileSizes.poll();
    pushSize(dirSize);
    if (maxDepthToDisplay<0 || maxDepthToDisplay >= fileSizes.size()) {
        System.out.println(level(fileSizes.size()) + dir + " " +␣
↪humanReadableByteCount(dirSize, true));
    }
    return FileVisitResult.CONTINUE;
}

@Override
public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws␣
↪IOException {
    if (Files.isRegularFile(file)) {
        pushSize(Files.size(file));
    }
    return FileVisitResult.CONTINUE;
}

@Override
public FileVisitResult visitFileFailed(Path file, IOException exc) throws IOException
↪{
    return FileVisitResult.CONTINUE;
}
```

Complete example

```
import java.io.IOException;
import java.nio.file.*;
import java.nio.file.attribute.BasicFileAttributes;
import java.util.ArrayDeque;
import java.util.Collections;
import java.util.Queue;
```

(continues on next page)

(continued from previous page)

```
public class PathExamples {

    private static class Visitor extends SimpleFileVisitor<Path>{

        long maxDepthToDisplay;

        Queue<Long> fileSizes = Collections.asLifoQueue(new ArrayDeque<>());

        protected Visitor(long maxDepthToDisplay) {
            super();
            this.maxDepthToDisplay = maxDepthToDisplay;
            fileSizes.add(0L);
        }

        private void pushSize(long size){
            long lastSize = fileSizes.poll();
            lastSize+=size;
            fileSizes.add(lastSize);
        }

        private String level(int depth){
            StringBuilder sbr = new StringBuilder();
            for (int ii = 0; ii < depth; ii++) {
                sbr.append(" ");
            }
            return sbr.toString();
        }

        /**
         * http://stackoverflow.com/a/3758880
         */
        public static String humanReadableByteCount(long bytes, boolean si) {
            int unit = si ? 1000 : 1024;
            if (bytes < unit) return bytes + " B";
            int exp = (int) (Math.log(bytes) / Math.log(unit));
            String pre = (si ? "kMGTPE" : "KMGTPE").charAt(exp-1) + (si ? "" : "i");
            return String.format("%.1f %sB", bytes / Math.pow(unit, exp), pre);
        }

        @Override
        public FileVisitResult preVisitDirectory(Path dir, BasicFileAttributes attrs)␣
↪throws IOException {
            fileSizes.add(0L);
            return FileVisitResult.CONTINUE;
        }

        @Override
        public FileVisitResult visitFileFailed(Path file, IOException exc) throws␣
↪IOException {
            return FileVisitResult.CONTINUE;
        }

        @Override
        public FileVisitResult postVisitDirectory(Path dir, IOException exc) throws␣
↪IOException {
            long dirSize = fileSizes.poll();
```

(continues on next page)

(continued from previous page)

```
            pushSize(dirSize);
            if (maxDepthToDisplay<0 || maxDepthToDisplay >= fileSizes.size()) {
                System.out.println(level(fileSizes.size()) + dir + " " +␣
↪humanReadableByteCount(dirSize, true));
            }
            return FileVisitResult.CONTINUE;
        }

        @Override
        public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws␣
↪IOException {
            if (Files.isRegularFile(file)) {
                pushSize(Files.size(file));
            }
            return FileVisitResult.CONTINUE;
        }
    }

    public static void main(String[] args) throws IOException {
        Path path = Paths.get(args[0]);
        Files.walkFileTree(path, new Visitor(3));

    }
}
```

11.1.2 Fork Join Framework

Java 1.7 has very nice Fork-Join Framework, that allows one to dynamically split between cores, but here is the catch: we don't know the amount of work needed upfront.

I have decided to try this framework, to (once again) summarize size of a directory tree.

This framework is nicely explained in the tutorials.

Overall I'm surprised with the performance of both naive and parralel implementation, naive version takes 6 seconds (when ran on my 150GB home directory), while parralel takes 3sec.

Code Highlights

Task result is a POJO object, containing path, it's size, information whether this path is a directory, and sub directories (if any). Here is the definition:

```
private static class WalkFileResult{
      public final Path dirPath;
      public final boolean isDir;
      public final long dirSize;
      public final List<WalkFileResult> subdirs;

      public WalkFileResult(Path dirPath, long dirSize) {
          this(dirPath, dirSize, Collections.emptyList());
      }

      public WalkFileResult(Path dirPath, long dirSize, List<WalkFileResult> subdirs)
↪{
          super();
```

(continues on next page)

(continued from previous page)

```
        this.dirPath=dirPath;
        this.dirSize=dirSize;
        this.isDir=Files.isDirectory(dirPath);
        this.subdirs=subdirs;
    }
}
```

Single task has following logic:

1. If we are looking at a file, calculate file size and return it.
2. If we are looking at a directory, create task for each child of the directory, execute these tasks in parrarel and then calculate the size.

In Java it is:

```
@Override
protected WalkFileResult compute() {
    try {
        if (Files.isRegularFile(currentPath)) {
            return new WalkFileResult(currentPath, Files.size(currentPath));
        } else if(Files.isDirectory(currentPath)) {
            List<WalkFileTask> subTasks = getSubtasks();
            return joinOnSubtasks(subTasks);
        }
    }catch (IOException | InterruptedException e){
        throw  new RuntimeException(e);
    }catch (ExecutionException e){
        throw new RuntimeException(e.getCause());
    }
    return new WalkFileResult(currentPath, 0L);
}

 private List<WalkFileTask> getSubtasks() throws IOException{
          // This visitor just returns immediate children of current path
          Visitor v = new Visitor(currentPath);
          Files.walkFileTree(currentPath, v);
          return v.subtasks;
      }

      private WalkFileResult joinOnSubtasks(List<WalkFileTask> subTasks) throws␣
↪ExecutionException, InterruptedException {
          long size = 0;
          List<WalkFileResult> subDirs = new ArrayList<>();
          for (WalkFileTask res: invokeAll(subTasks)){
              WalkFileResult wfr = res.get();
              size+=wfr.dirSize;
              if (wfr.isDir){
                  subDirs.add(wfr);
              }
          }
          return new WalkFileResult(currentPath, size, subDirs);
      }
```

Complete example

```
package examples;

import javax.sound.midi.SysexMessage;
import java.io.IOException;
import java.nio.file.*;
import java.nio.file.attribute.BasicFileAttributes;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.ForkJoinTask;
import java.util.concurrent.RecursiveTask;



public class ForkJoinPath{

    private static class WalkFileResult{
        public final Path dirPath;
        public final boolean isDir;
        public final long dirSize;
        public final List<WalkFileResult> subdirs;

        public WalkFileResult(Path dirPath, long dirSize) {
            this(dirPath, dirSize, Collections.emptyList());
        }

        public WalkFileResult(Path dirPath, long dirSize, List<WalkFileResult>␣
↪subdirs) {
            super();
            this.dirPath=dirPath;
            this.dirSize=dirSize;
            this.isDir=Files.isDirectory(dirPath);
            this.subdirs=subdirs;
        }
    }

    private static class WalkFileTask extends RecursiveTask<WalkFileResult>{

        private static class Visitor extends SimpleFileVisitor<Path>{

            public final Path root;

            public List<WalkFileTask> subtasks = new ArrayList<>();

            protected Visitor(Path root) {
                super();
                this.root = root;
            }

            @Override
            public FileVisitResult preVisitDirectory(Path dir, BasicFileAttributes␣
↪attrs) throws IOException {
                if(Files.isSameFile(dir, root)){
```

(continues on next page)

(continued from previous page)

```
                    return FileVisitResult.CONTINUE;
                }
                if (Files.isReadable(dir)) {
                    subtasks.add(new WalkFileTask(dir));
                }
                return FileVisitResult.SKIP_SUBTREE;
            }

            @Override
            public FileVisitResult visitFile(Path file, BasicFileAttributes attrs)␣
↪throws IOException {
                if (Files.isReadable(file) && Files.isRegularFile(file)) {
                    subtasks.add(new WalkFileTask(file));
                }
                return FileVisitResult.CONTINUE;
            }

            @Override
            public FileVisitResult visitFileFailed(Path file, IOException exc) throws␣
↪IOException {
                return FileVisitResult.CONTINUE;
            }
        }

        private final Path currentPath;

        public WalkFileTask(Path currentPath) {
            super();
            this.currentPath=currentPath;
        }

        private List<WalkFileTask> getSubtasks() throws IOException{
            // This visitor just returns immediate children of current path
            Visitor v = new Visitor(currentPath);
            Files.walkFileTree(currentPath, v);
            return v.subtasks;
        }

        private WalkFileResult joinOnSubtasks(List<WalkFileTask> subTasks) throws␣
↪ExecutionException, InterruptedException {
            long size = 0;
            List<WalkFileResult> subDirs = new ArrayList<>();
            for (WalkFileTask res: invokeAll(subTasks)){
                WalkFileResult wfr = res.get();
                size+=wfr.dirSize;
                if (wfr.isDir){
                    subDirs.add(wfr);
                }
            }
            return new WalkFileResult(currentPath, size, subDirs);
        }

        @Override
        protected WalkFileResult compute() {
            try {
                if (Files.isRegularFile(currentPath)) {
                    return new WalkFileResult(currentPath, Files.size(currentPath));
```

(continues on next page)

(continued from previous page)

```
                } else if(Files.isDirectory(currentPath)) {
                    List<WalkFileTask> subTasks = getSubtasks();
                    return joinOnSubtasks(subTasks);
                }
            }catch (IOException | InterruptedException e){
                throw  new RuntimeException(e);
            }catch (ExecutionException e){
                throw new RuntimeException(e.getCause());
            }
            return new WalkFileResult(currentPath, 0L);
        }
    }

    private static String level(int depth){
        StringBuilder sbr = new StringBuilder();
        for (int ii = 0; ii < depth; ii++) {
            sbr.append(" ");
        }
        return sbr.toString();
    }

    /**
     * http://stackoverflow.com/a/3758880
     */
    public static String humanReadableByteCount(long bytes, boolean si) {
        int unit = si ? 1000 : 1024;
        if (bytes < unit) return bytes + " B";
        int exp = (int) (Math.log(bytes) / Math.log(unit));
        String pre = (si ? "kMGTPE" : "KMGTPE").charAt(exp-1) + (si ? "" : "i");
        return String.format("%.1f %sB", bytes / Math.pow(unit, exp), pre);
    }


    private static void printResult(WalkFileResult wfr, int depth){
        if (depth >= 3){
            return;
        }

        System.out.println(level(depth) + wfr.dirPath + " " +␣
↪humanReadableByteCount(wfr.dirSize, false));
        for (WalkFileResult child: wfr.subdirs){
            printResult(child, depth+1);
        }



    }

    public static void main(String[] args) throws ExecutionException,␣
↪InterruptedException {
        long start = System.nanoTime();
        Path path = Paths.get(args[0]);
        ForkJoinPool pool = new ForkJoinPool();
        WalkFileTask task = new WalkFileTask(path);
        pool.execute(task);

        printResult(task.get(), 0);
        double duration = (System.nanoTime() - start) * 1E-9;
```

(continues on next page)

(continued from previous page)

```
        System.out.println(duration);



    }
}
```

11.1.3 Notable mentions

There is also very nice `WatchService`, that allows to monitor filesysem for file changes.

11.2 Java 1.8

11.2.1 Streams and Lambdas

Third attempt to do the same task: to summarize size of a directory tree.

This times using `Streams` and `Lambdas`. Solution is most concise, but least readable IMO. Also, while other solutions transparently handle unreadable directories, this one explodes with AccessDenied exception.

Code Highlights

Result POJO:

A function that can throw an exception:

```
@FunctionalInterface
public interface CheckedFunction<T, R> {
    R apply(T t) throws IOException;
}
```

A lambda that calculates file size:

```
CheckedFunction<Path, DirSize> mapper = (Path p) -> new DirSize(p,
    Files.walk(p).parallel()
    .filter(Files::isReadable)
    .mapToLong(StreamExamples::safeSize).sum());
```

A stream that walks over FS calculating size of each directory:

```
Files.walk(path, 3)
  .parallel().filter(Files::isDirectory).filter(Files::isReadable).map(
      (Path p) -> {
          try {
              return mapper.apply(p);
          } catch (IOException e) {
              return new DirSize(p, -1);
          }
      }).forEach(
        (DirSize d) ->
          System.out.println(d.path + " " + humanReadableByteCount(d.size, false)));
```

Complete example

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

/**
 * Created by jb on 12/3/15.
 */
public class StreamExamples {

    private static class DirSize{
        public final Path path;
        public final long size;

        public DirSize(Path path, long size) {
            this.path = path;
            this.size = size;
        }
    }

    @FunctionalInterface
    public interface CheckedFunction<T, R> {
        R apply(T t) throws IOException;
    }

    public static long safeSize(Path p){
        try {
            return Files.size(p);
        } catch (IOException e) {
            return 0;
        }

    }

    public static String humanReadableByteCount(long bytes, boolean si) {
        int unit = si ? 1000 : 1024;
        if (bytes < unit) return bytes + " B";
        int exp = (int) (Math.log(bytes) / Math.log(unit));
        String pre = (si ? "kMGTPE" : "KMGTPE").charAt(exp-1) + (si ? "" : "i");
        return String.format("%.1f %sB", bytes / Math.pow(unit, exp), pre);
    }

    public static void main(String[] args) throws IOException {
        long start = System.nanoTime();
        Path path = Paths.get(args[0]);

        CheckedFunction<Path, DirSize> mapper = (Path p) -> new DirSize(p,
                Files.walk(p, Integer.MAX_VALUE).parallel().filter(Files::isReadable).
↪mapToLong(StreamExamples::safeSize).sum());

        Files.walk(path, 3)
            .parallel().filter(Files::isDirectory).filter(Files::isReadable).map(
                (Path p) -> {
                    try {
                        return mapper.apply(p);
```

(continues on next page)

(continued from previous page)

```
                    } catch (IOException e) {
                        return new DirSize(p, -1);
                    }
                }).forEach((DirSize d) -> System.out.println(d.path + " " +␣
↪humanReadableByteCount(d.size, false)));

        double duration = (System.nanoTime() - start) * 1E-9;
        System.out.println(duration);
    }


}
```

CHAPTER 12

How to disable subpixel rendering in Jetbrains products

If you want to disable subpixel rendering (another term is font anitaliasing) in any JetBrains product (like: Idea, Pycharm and so on), you need to:

1. Go to `bin` dir of install folder, find file that ends with `*vmoptions` (`idea.vmoptions`, `pycharm.vmoptions`).
2. There should be a line `-Dawt.useSystemAAFontSettings=lcd` in this file, delete this line.
3. Add line: `-Dprism.lcdtext=false`.

These two lines define properties passed to Java Virtual Machine that control antialiasing, both of them are undocumented, so they might not work in future JVM releases (I tested on `1.8.51`). These JVM settings seem to clash with either JetBrains settings (that override these) and Gnome settings, so disabling them makes sense.

CHAPTER 13

Linux dynamically assigned ports

So you want to assign your application a `high` port number, maybe you do some testing (and you'll launch many instances of servers), maybe you want to set some service on unusual port (which is ok in some cases).

What you don't want is your selected port to clash with the range of ports that are automatically assigned to outgoing connections. This range is sadly OS dependent, IANA defines it as: 49152–65535, however linux chooses different range, that can be read by: ‘‘cat /proc/sys/net/ipv4/ip_local_port_range ‘‘.

For more information see this SO post.

CHAPTER 14

Asyncio servers in Python

From what I read: "normal" architecture for a web-server is that one assings single thread to a client, this is mostly OK, but after certain number of clients your performance drops (because of thread memory overhead, context switching costs, and other things). Specific limit is hart to guess, and depends on OS, app, hardware and so on.

Asynchronous servers were supposedly a solution to this problem, but I didn't really believe this. From what I understood you could always buy more frontend boxes and scale it this way.

But today I actually written a async server, this server is very simple, has no error control (it took me less than an hour, including reading manuals), protocol is very simple:

0. Protocol is line based
1. When any client sends a line of text to the server
2. This line is sent to every another client.

It was written in Python 3.5, server uses PEP-492 and the `asyncio` library. Go ahead and read this PEP, there is even a working example (that I based upon).

I didn't do extensive tests, but it seems that this server handles 4000 connections **easily** on single core, throughput is about 40K messages per second (still on single core).

Note: I wouldn't rely on these numbers much, to do a proper tests I would have to:

1. Add error handling to the server.
2. Change the protocol to acknowledge results.
3. Write some proper tests.

Tests are based on running 4 processes each spawning 1000 sockets to the server, and then writing one message per 100ms in yet another thread. Results were observed by looking at the output :)

Server works as follows:

1. Keeps a set of all connections (sockets in essence)
2. Each relieved line is sent to all sockets

Server code is here:

```
# -*- coding: utf-8 -*-

import asyncio
import threading
from asyncio.streams import StreamReader, StreamWriter

import time

from multiprocessing import Process, Lock, Condition


class AsynChat(object):

  def __init__(self, port, client_may_end_connection=False):
    self.clients = set()
    self.port = port
    self.client_may_end_connection=client_may_end_connection
    self.__loop = None

  async def handle_connection(self, reader:StreamReader, writer:StreamWriter):
    self.clients.add(writer)
    while True:
      data = await reader.readline()
      print(data.strip())
      if self.client_may_end_connection and data.strip() == b'END':
        print("STOP")
        self.__loop.stop()
      for w in self.clients:
        if w == writer:
          continue
        w.write(data)
        await w.drain()
      if not data:
        if writer in self.clients:
          self.clients.remove(writer)
          try:
            writer.write_eof()
          except OSError:
            pass # Sometimes it explodes if socket was closed very soon, didn't␣
↪investigate
          return

  async def echo_server(self):
      await asyncio.start_server(self.handle_connection, 'localhost', self.port)

  @classmethod
  def run_in_process(cls, *args, **kwargs) -> Process:
    c = Condition(Lock())
    def build_and_run(*args, **kwargs):
      ac = cls(*args, **kwargs)
      ac.run_loop(c)

    p = Process(target=build_and_run, args=args, kwargs=kwargs)
    p.start()
    with c:
      c.wait()
```

(continues on next page)

(continued from previous page)

```
    return p

  def run_loop(self, c:Condition=None):
    self.__started = True
    self.__loop = asyncio.get_event_loop()
    self.__loop.run_until_complete(self.echo_server())

    def notif():
      with c:
        c.notify()

    try:
      if c:
        self.__loop.call_soon(notif)
      self.__loop.run_forever()
    finally:
      self.__loop.close()



if __name__ == "__main__":
  a = AsynChat(1234, True)
  a.run_loop()
```

CHAPTER 15

How to discard connection in a fast way

Just a quick note, here is how to drop a TCP connection fast:

- Client sends `SYN` packet
- Server responds with `RST` packet, which promptly kills the connection without any state.

CHAPTER 16

Some remarks about databases

Note: I was refreshing my knowledge on the broad subject of databases, mostly by reading Wikipedia articles, which resulted (apart from notes attached here) in creating this wikipedia book (in case some future problems with Wikipedia `you can try version hosted here`).

16.1 Read anomalies Anomalies

What can happen if your database doesn't serialize transactions properly:

Dirty read When one transaction sees uncommitted data of another one

Short example:

We have two transactions T1 and T2.

- T1 reads value of a row R
- T2 writes new value of row R
- T1 sees a change in row value

Non-Repeatable Read When value of a row changes during a transaction.

Short example:

We have two transactions T1 and T2.

- T1 reads value of a row R
- T2 writes new value of row R
- T2 commits <- **Difference from Dirty Read**
- T1 sees a change in row calue

Phantom read When a result of query changes during the transaction.

> **Note:** Database without Non-Repeatable Reads cam still suffer from phantom reads, non-repeatable reads apply when other transaction updates or deletes records while phantom also apply to inserted rows.

Short example:

- T1 executes a query: `SELECT AVG(value) FROM account WHERE ...GROUP BY ...`
- T2 executes: `INSERT .... INTO account`
- T2 commits
- T1 executes a query: `SELECT AVG(value) FROM account WHERE ...GROUP BY ...` and gets different results.

Write Skew This is an anomaly for MVCC databases, which occours each transaction works on a snapshot of database, and don't see changes done by each other.

> **Note:** This anomaly is not defined in SQL standard, as SQL standard had locking databases in mind.

Short example:

Let's consider a banking application, with following constraint: balance of account must be non-negative, but we don't store it explicitly, it is just calculated by `SELECT SUM(t.value) FOR account .... INNER JOIN transaction as t ON ...`

- At the start of the transaction balance of account A is 100PLN
- Transaction T1 starts
- Transaction T2 starts
- Transaction T1 adds a transaction for A that withdrawns 100PLN (which is valid as constraint is held)
- T1 Commits
- Transaction T2 does the same (which still is valid as constraint is held inside a snapshot for T2 — constraint is not held "outside of" this snapshot).

> **Note:** Wikipedia says that you might resolve this issue by explicitly writing to a dummy row just to force write-write confilct.

16.2 Transaction isolation Levels

Serializable Highest transaction isolation. Transactions are executed *as if* they were executed serially.

No read anomalies can occur.

In database that uses locks it requires to put locks on:

- Every row you write to
- Every row you read
- Range lock for every query (for example lock all records for which specified condition is true).

These locks are held until the end of the transaction.

Snapshot Isolation This transaction isolation level works as follows: Each transaction sees database in a (consistent) state the database was at the beginning of the transaction (with any changes it did).

This is very different from Serializable, as it allows "Write Skew" anomalies.

Note: This isolation level is not defined in SQL standard.

You can define Serializable isolation on top of Snapshot Isolation relatively easy by:

- This can be done relatively easy by the DMBS, see this article:

  Cahill, M. J., Röhm, U., & Fekete, A. D. (2009). Serializable isolation for snapshot databases. ACM Transactions on Database Systems, 34(4), 1–42. doi:10.1145/1620585.1620587

- Writing proper code that introduces artificial write conflicts between data.

  Note: These conflicts are artificial, because they exist only to introduce write conflicts, which will abort transaction that tries to write to the database as the second one.

Repeatable Read This isolation level reading a row will always produce the same value, even of these rows were changed by other transactions. However query results are can change during the transaction (especially for aggregate queries).

In database that uses locks it requires to put locks on:

- Every row you write to
- Every row you read

These locks are held until the end of the transaction.

Read Committed In this isolation level transaction doesn't see changes made by another uncommitted transactions, however it may see changes made by commited transactions.

In database that uses locks it requires to put locks on:

- Every row you write to
- Every row you read

Write locks are held until the end of the transaction, however read locks are released after each select.

Read Uncommitted In this level you can see uncommitted data sent by saved by other transactions.

16.3 Serializability

We have a some set of concurrent transactions, these transactions are serializable, if one can produce a schedule containing these transactions executed serially in some order.

Serializability is important because:

If DBMS checks if database is in a consistent state **after each of the transactions**, and transactions are serializable — it means that the database is in a consitent state after all transactions.

CHAPTER 17

MVCC in postgresql

Note: This is based on my lecture on the databases, which (is still available in Polish)

There are two ways to implement proper transaction isolation:

- First is by using locking. In this padagrim whenever transaction reads data a lock is issued, and any write to that data will wait until reading transaction finishes (this might be simplified).
- Second is by using MVCC — that is multi version concurrency. It works as follows: each transaction sees database in a state **at the time the transaction**, so reads and writes don't need to wait for each other (there is a problem with write skew anomaly, which is solved by the postgresql 9.1 and newer.

17.1 How does MVCC work

17.1.1 Start of the transaction

When transaction starts following things happen (or may happen depending on isolation level):

- Transaction is assigned a `txid` a transaction ID, transaction id's are ordered 32 bit integers (that may wrap around at some point in time, but Postgres handles it).
- `txid``s of all committed transactions are stored (possibly in a more efficient way than storing all ``txids)`

17.1.2 Data constraints

Each row contains couple of magic columns:

`xmin` This is the `txid` of transaction that inserted this row

`xmax` This is the `txid` of transaction that deleted this column

cmin, cmax Index of statement in transaction that added/deleted that row

17.1.3 Basic operations

INSERT When a transaction inserts a row it `xmin` is set to `txid` of this transaction.

DELETE When a transaction deletes a row it just sets `xmax` to it's `txid`

UPDATE Updates are replaced with a delete and insert pair.

17.1.4 Data visibility

Row is visible for transaction `txid` if (all statements must be true):

- It's `xmin < txid` (row was inserted by a transaction before this one).
- Transaction `xmin` is commited (in case of Read Commited isolation level), or `xmin` was commited before start of current transaction (other isolation levels)
- It's `xmax` is empty or `xmax > txid` (row was deleted by a transaction that started after this one).

In case of transaction that issue multiple statements `cmin`, `cmax` are used for example to have a cursor that consistently iterates over a table, even if the same transaction alters the table.

17.2 VACUUM

Data can't be deleted from disk immediately in databases using MVCC, because ongoing transactions might not 'see' the delete, and still need to access deleted row. Data can be deleted only after all transactions with `txid` lower row's than `xmax` have either committed or have been rolled back.

Postgresql does this (more or less) automatically, but you might call `VACUUM` by hand if you need to reclaim space (this space will not necessarily be freed to the OS, rather it will be accessible for new inserts).

17.3 Sources

- BRIUCE MOMJIAN `MVCC Unmasked`: http://momjian.us/main/writings/pgsql/mvcc.pdf (I have also cached it `locally`, due to permissive CC license)
- Serializable Snapshot Isolation on Postgresql Wiki, (due to even more permissive license it is also cached `locally`.
- My old lectures: http://db.fizyka.pw.edu.pl/~bzdak/bazy_danych_ed_20/wyklad10/wyk10.html#mvcc-w-postgresq
- Wikibook I have collected (in case some future problems with Wikipedia `you can try version hosted here`).

CHAPTER 18

Should you use BRTFS for your laptop?

TL; DR; Probably not.

Recently I re-installed by Debian desktop, and enabled full disk encryption, (and LVM for that matter). I was also toying with the idea of using BTRFS, which has many features I always wanted to have, including:

- Fast fs-level compression
- Optional, out of write path, deduplication.
- Very nice features including super-easy resizing (adding more GBs to your file system takes seconds)
- Copy on write semantics
- File system snapshoting (said to be good for backups)

BTRFS is stable, yet I believe is unusable for a most of people. If you are not a linux nerd, don't try, if you are you might, but remember:

- Do backups, you should always do backups, especially if you use encrypted filesystems on ssd drives.
- Do yourself a favour and buy an USB stick, and burn there a linux live-cd.
- BTRFS **will** suprise you.

Here are two nasty suprises I had.

18.1 BTRFS COW doesnt play well with some usage patterns

If you are a linux nerd, you probably have some virtual machines, COW (copy on write) doesn't play with them. Well everywhere where you have large files that are written to often, it doesn't play well with COW.

COW can be disabled on directory level, do to this issue *chattr +C /dir* command, this will disable COW for everything under */dir*. Keep in mind that it works **only on empty files and directories**, turning off COW on a file with data, has **undefined behaviour**, and most often is bad. Turning off COW for directories with files is safe, but existing files will have COW enabled.

In my case VirtualBox failed with very non-obvious errors, before I disabled COW.

18.2 BTRFS needs garbage collection (or something similar)

Basically BTRFS kind-of lies to the OS when reporting free space. You can have full filesystem and yet BTRFS will report 100GB of free space. I don't try to understand what it is, BTRFS wiki says things like: "The primary purpose of the balance feature is to spread block groups across all devices so they match constraints defined by the respective profiles."

Usable free space on your disk can be seen using *btrfs fi show* Which will display: total system size, and how much space is currently used by btrfs.

In my case:

```
# btrfs fi show

Label: none  uuid:
    Total devices 1 FS bytes used 613.29GiB
    devid    1 size 745.06GiB used 649.06GiB path /dev/mapper/
```

I have 745GB partition, of which 649GB is used by btrfs, however, only 613 is used for files.

This can be fixed by issuing something like:

```
btrfs balance start -dusage=<magic number> / &
```

<<magic number> is a percentage value, and BTRFS will try to "rebalance" only chunks filled in less than this <<magic number>>, the bigger number you put there the longer will it take, and the more space you'll reclaim. You can start with percentage value of used space you have on your drive.

CHAPTER 19

How to deploy django application on debian server (UWSGI version)

19.1 Install proper version of python

On debian I'd discourage using `system` python for deployment — mostly becaluse they tend to upgrade minor python versions without notice, which sometimes breaks `C ABI` in installed virtualenvs.

So either roll your own `deb` files that install pythons somewhere in `/usr/local` or compile python on server (if you frown on having development tools on your server roll `debs`).

Note: For development environment Pythonz is a nice tool to compile (and manage) many versions of python.

If you want deploy your server by hand just download and compile python.

If you are into automatic deployment (and you **should** be)

19.2 Assumptions about the system

I'll assume that you will configure your system in following way:

- Django application will be using `www-client` user
- Code will be inside `/home/www-client/repo`
- There will be a django generated uwsgi file in `/home/www-client/repo/webapp.uwsgi`
- Virtualenv will be in `/home/www-client/repo/venv`.

19.3 Install your application into virtualenv

You know how to do that don't you?

Now you can tests whether your setup is correct, just run

```
./manage.py runserver
```

and see if you can connect to your application.

19.4 Install uwsgi into your virtualenv

Install `uwsgi` **into your virtualenv** from `pip`. Now you can run your application using uwsgi:

```
uwsgi --http :8000 --module webapp.wsgi
```

I strongly discourage you from using uwsgi bundled with system.

You can configure `uwsgi` using a variety of ways, most of which are better than using a command line arguments :). For example you can create an ini file named `uwsgi.ini`:

```
module=webapp.wsgi
pythonpath=/home/www-client/webapp
http=8000
```

And then start uwsgi using: `uwsgi --ini uwsgi.ini`.

19.5 Use systemd to launch your applicaiton

Now use systemd to launch the application.

Systemd is a very nice init system that is becoming a standard in most recent distributions (it's even on Debian stable).

If your distribution has no systemd you can use supervisord (which is even on debian oldstable), tutorial to deploy django with it *is here*.

So create `/home/webapp/uwsgi.ini` file with following contents:

```
module=webapp.wsgi
http=8000
pythonpath=/home/www-client/repo
```

Create a `webapp.service` file and put it in `/etc/systemd/system/`, file should have following contents:

```
[Unit]
Description=Description
After=syslog.target

[Install]
WantedBy=multi-user.target

[Service]
# What process to start
ExecStart=/home/www-client/venv/bin/uwsgi --ini /home/www-client/uwsgi.ini
# What user chown to
User=www-client
# Working directory
WorkingDirectory=/home/www-client/webapp
Restart=always
```

(continues on next page)

(continued from previous page)

```
# Kill by SIGQUIT signal --- this is what asks wsgi to die nicely
KillSignal=SIGQUIT
# Notify type, in this type uwsgi will inform systemd that it is ready to handle␣
↪requests
Type=notify
StandardError=syslog
NotifyAccess=all
```

Then:

```
sudo systemctl --system enable webapp
sudo systemctl start webapp
```

Now you should have a working uwsgi configuration, which is reachable at `localhost:8000`.

19.6 Connect `uwsgi` and `nginx`

Note: This part is more or less ripoff from: http://uwsgi-docs.readthedocs.org/en/latest/tutorials/Django_and_nginx.html

In this part we will put `uwsgi` behind `nginx` server.

Add following sections to nginx configuration:

```
upstream django {
    # server unix:///path/to/your/mysite/mysite.sock; # for a file socket
    server 127.0.0.1:8000; # for a web port socket (we'll use this first)
}

# configuration of the server
server {
    # the port your site will be served on
    listen      80;
    # the domain name it will serve for
    server_name .example.com; # substitute your machine's IP address or FQDN
    charset     utf-8;

    # max upload size
    client_max_body_size 75M;   # adjust to taste


    # Finally, send all non-media requests to the Django server.
    location / {
        uwsgi_pass  django;
        include     /etc/nginx/uwsgi_params; # the uwsgi_params file you installed
    }
}
```

Now you should see something on your server port `80`. To finalize our setup we need to create static and media directories.

19.7 Connect ngingx and uwsgi via linux file sockets

Because of many (peformance, safety) reasons it is better to connect nginx with uwsgi via linux domain sockets.

First replace `uwsgi.ini` file with something like:

And also create `/home/www-client/sock/`.

This does the following: creates a socket in `$HOME/sock/webapp.sock`, sets its group ownership to `www-data` (which is user/group used by both `nginx` and `apache` on default debian configuration).

Note: Linux file sockets use normal file permissions, so `nginx` has to have read-write access to it.

Then replace:

```
upstream django {
    server 127.0.0.1:8001; # for a web port socket (we'll use this first)
}
```

with:

```
upstream django {
    server unix:///path/to/your/mysite/webapp.sock; # for a file socket
}
```

19.8 Update media and static root

Now we'll update settings so static media is served by nginx.

19.8.1 Update settings py

You'll need to update `STATIC_ROOT`, `STATIC_URL`, `MEDIA_ROOT`, `MEDIA_URL` settings of youe app.

Something along the lines:

```
MEDIA_ROOT = '/var/drigan-media'
MEDIA_URL = '/media/'
STATIC_ROOT = '/var/drigan-static'
STATIC_URL = '/static/'
```

19.8.2 Update nginx

```
location /media  {
    alias /path/to/your/mysite/media;  # your Django project's media files - amend as␣
↪required
}

location /static {
    alias /path/to/your/mysite/static; # your Django project's static files - amend␣
↪as required
}
```

19.9 Tweak uwsgi so it scales

You might want tweak `uwsgi` so it launches more processes/workers, but this well outside scope of this tutorial.

CHAPTER 20

How to encrypt a usb drive using cryptsetup

Actuall I didn't manage to set up it using console `cryptsetup` commands, however there is a very nice dommand `gnome-disks` (which comes in package `gnome-disk-utility`) that does everything automatically!

CHAPTER 21

Disk performance

I'm doing a small side project, that requires me to randomly query/update about 1TB of data. Managing random access to this kind of data is non trivial.

21.1 Attempt 1 or the problem

So I took 2TB hard disk out of my drawer and tried to use it.

Lessons learned:

21.1.1 Lesson 1: Don't use BTRFS for data volumes

Don't use BTRFS for data volumes (at least not without serious tweaking: for starters you need to disable COW for data volumes).

On the other hand: BTRFS is totally cool fs for daily use, and has native support for compression and native out of band deduplication.

I decided to use `xfs` which is a nice tweakable file system.

21.1.2 Lesson 2: There are some kernel settings that require tweaking

Kernel is not really tweaked for serious disk IO initially.

Especially following settings are critical:

- `/proc/sys/vm/dirty_ratio` or `/proc/sys/vm/dirty_bytes`. Both control maximal amount of dirty pages that system can hold. If any process dirties more pages, writes will be stopped until enough bytes are written do disk so we finish before the threshold.

  If some process sends writes a lot of data, and then issues `fsync` system may become unresponsive. (This is what inserting a lot of data into SQL database does).

- `/proc/sys/vm/dirty_background_bytes` or `/proc/sys/vm/dirty_background_ratio`

  If there are more dirty pages than this threshold system starts background process to write some data to disk.

`_ratio` variables are older, and are mentioned everywhere. They basically mean if more than `N` percent of RAM is used as dirty cache then flush.

`_bytes` variables were added "recently" and often they are not mentioned on tutorials. They are easier to understand, and are a better fit for systems with a lot of RAM. They just mean threshold in bytes.

Default values are ridiculously high on systems with a lot of RAM – as by default `_ratio` variant is used with setting of about 10 od 20, which may mean that your system will flush gigabytes of data on `fsync`.

This values should be much lower than defaults. You need to measure this (this is one thing I learned much later)

Easiest way to set these variables is to add following to `rc.local`.

21.1.3 Lesson three: set noatime to your data partition

`noatime` is a mount option that disables `last access time` attribute on files. However this attribute might be useful, it turns every file read into a write (to update this attribute) which is a big no.

To do this add `noatime` to appropriate `/etc/fstab` entry:

```
/dev/mapper/..      /mount-point xfs defaults,noatime 0 0
```

21.1.4 Lesson four: Read a book

I recommend: `PostgreSQL High Performance`. If you know any books on tuning IO devices I'd be happy to read them (so e-mail me)

21.1.5 Lesson five: IOPS is an important parameter

Iops stands for "IO Operations Per Second", or "how many random reads" your disk can make in a second.

Nice thing about SSD's is that they mostly allow random access to data — that is it doesn't matter much whether you read one big continoous file or just read random parts of said file.

HDD's don't have this property — if you read data sequentially you'll have much better read speeds (orders of magnitude better).

So for HDD's IOPS are limiting factor in case of random access.

To see how much iops is your disk currently using install `sysstat` Debian package and user `iostat`:

```
Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sdc             113.53
```

and `tps` is column you need.

Typical number of IOPS you can get from HDD is `15-50` for `5400 RPM` disks, and `50-100` for `7200 RPM` disks (according to Wikipedia). `iostat` showed more IOPS for my `5400 RMP` disk which might be caused by on-disk cache.

21.1.6 Problem

I just didn't get enough IOPS.

21.2 Non solutions

21.2.1 Use cloud

Easiest solution would be just do to what everybody does: "just put it in the cloud". The problem is it's expensive, and I'm stingy. Probably there will be no money from this project, and if I can just scrap it from parts that lie in my drawer why spend money.

For now let's ignore VM costs, and focus on the hardware:

- On AWS I could buy 1TB of SSD storage for 119$ a month "throughput optimized

HDD" for 50$. SSD would be way faster than my solution, and I believe HDD's would be slower. * On Google Cloud both HDD and SSD would be faster than my solution as google's HDD's are not that throughput oriented. SSDs cost 200$ and HDDs 50$.

If I add VM to this I get slightly about 200$ a month.

Note: I'm totally for using cloud for deploying your services, managing hardware that has couple of nines availability is not trivial, and well worth the price.

21.2.2 Just buy SSDs

Just buy SSD disks. This would work, but again: I'm stingy and 2 TB SSD disk is about 850$ (and this is not one of my favourite makes).

If I was sure total data will be less than 1TB I'd probably buy 1TB SSD, but prices of SSD's skyrocket after 1 TB in size.

21.3 Attempt 2

I bought two 2TB 7400 RPM Western Digital hard disks. I'm a big fan od WD disks.

And I decided to just use them as some form of `RAID-0`. This way I'll get 4TB of total storage with decent speeds (but if any drive dies I loose my data).

21.3.1 Lesson 0: Raid is not a backup

This is not really something I encountered right now, but I feel this can't be stressed enough.

Note: I also dislike using RAID controllers — as there is no such thing as `raid disk format` and each manufacturer might make their own disk format, which means that when controller dies your data might be hard to recover.

21.3.2 Lesson 1: Raid is everywhere

In linux at least you might get RAID or RAID-like behaviour on many different levels:

- RAID controller
- Software RAID
- LVM

- Filesystems — this is totally cool that you can have native `RAID-X` in your BTRFS system on **file system level**.

21.3.3 Lesson 2: LVM striping is cool

If you have more than single disk in your LVM volume group, you can use striping — works more or less like `RAID-0` but with some advantages:

- It's easy to set up.
- You can easily create and delete volumes with and without striping.
- You don't have this assumption that: "size of volume is lowest of the size of physical volumes".

Downside is that probably Raid is gives you better performance (at least this is what I read on the Internet)!

21.3.4 Lesson 3: Stripe size matters

I basically created striped volume on two disks, and I got no performance increase whatsoever. Synthetic tests showed performance **decrease**, yes: two `7400RPM` disks got slower than single `5400RPM` disk.

21.3.5 Lesson 4: Measure performance

There are very nice performance tools. On I used was `sysbench` (it's in debian repositories).

My initial assumption is that I selected wrong stripe size. Stripe is amount of contiguous data that LVM puts on a single disk, next stripe is on next disk and so forth.

There is no "best" stripe size, I would suggest having a stripe larger than your typical random read — so typical random read will not require coordinating couple of disks.

21.3.6 Lesson 5: How to measure performance

Currently I did very simple benchmark:

1. I created a volume with given stripe size
2. I created a XFS filesystem with block size equal to stripe size (not sure if this is good idea)
3. Prepared `sysbench` by: `sysbench fileio --file-total-size=128GB prepare`
4. Ran `sysbench fileio --file-total-size=128GB --file-test-mode=rndrw --file-fsync-freq=10 --threads=8` for couple of block sizes (block size is how much data is read or written in single batch)

If you can read python, here is script I used: :download:`data/striped-benchmark.py`.

My results were as follows:

- For larger block sizes bigger stripes gave me much more throughput.
- For smaller block sized bigger stripes were not that important.

Ultimately I decided I'll use `64kb` stripe size.

Right now I get about 100 IOPS for each drive and more than 100mb/s read and write speeds from both drives (in my specific workload!).

Which probably is enough for time being, and I'm not sure I could get better performance from two disks that costed less than 300$ (or 1000 złotys).

If I'll do some tweaking I'll let you know what and how I did it.

Comments:

- If you have any comments please send me an e-mail to `jacekfoo@gmail.com`.

CHAPTER 22

Indices and tables

- genindex
- modindex
- search