

POLITECHNIKA WROCŁAWSKA

APLIKACJE INTERNETOWE I ROZPROSZONE

Generowanie fraktali z użyciem klastra obliczeniowego

Jakub Burzała

Krzysztof Cabała

Bartosz Cieśla

Adrian Frydmański

Dawid Gracek

Bartosz Kardas

pod przewodnictwem
dr hab. inż. Henryka MACIEJEWSKIEGO

2 czerwca 2016

Spis treści

1	Cel projektu i wymagania	2
2	Prototyp	4
2.1	Opis ogólny	4
2.2	Opis szczegółowy	4
3	System w wersji finalnej	5
3.1	Baza danych	5
3.2	MPI	6
3.2.1	Master	6
3.2.2	Slave	8
3.2.3	Mandelbrot – generowanie fraktala	9
3.3	Django	10
3.4	Strona internetowa – interfejs użytkownika	16
4	Uruchamianie	23
5	Testy wydajności	23
6	Podsumowanie	27

1 Cel projektu i wymagania

Celem projektu jest stworzenie aplikacji do generowania fraktali z użyciem technologii internetowych i zrównoległonych obliczeń na klastrze MPI.

Użytkownik aplikacji powinien móc:

1. Wpisać dowolną funkcję zespoloną generującą fraktal
2. Zobaczyć wygenerowany fraktal
3. Przybliżyć obraz w pewnym punkcie płaszczyzny
 - klikając
 - suwakiem
 - zaznaczając pewien obszar
4. Określić:
 - czas trwania animacji
 - liczba klatek na sekundę
 - rozdzielczość
5. Zatrzymać, przewinąć, wznowić animację
6. Wybrać rodzaj animacji. Animowanie względem:
 - przybliżenia – wartość początkowa i końcowa
 - punktu centralnego (ścieżki) – wybranie dwóch punktów na płaszczyźnie (początek i koniec) oraz zdefiniowanie ścieżki pomiędzy nimi, która wskazuje w jaki sposób będzie poruszać się kamera
 - parametrów zespolonych – dowolność we wprowadzaniu parametrów w równaniu zespolonym. Każdy z nich może być zmieniany z zadany krok podczas animacji
 - kolorystyki – wybór kolorystyki fraktala, definiowanie własnych zasad kolorowania
 - kroków zbieżności – ilość maksymalnej ilość kroków zbieżności, potrzebnych do obliczenia koloru danego piksela
7. Łączyć powyższe animacje:
 - szeregowo – animacje odtwarzają się po sobie
 - równolegle – animacje wykonują się w tym samym momencie
8. Zobaczyć postęp prac przy generacji animacji w postaci paska postępu oraz przewidywany czas zakończenia
9. Zobaczyć skalę przybliżenia w postaci łatwej do wyobrażenia jednostki. Przykładowo punkt początkowy: 100 px odpowiada 1 km. W miarę przybliżania aktualizowanie jednostki do m, cm, mm itd.
10. Zapisać opis animacji (funkcja, parametry itp) na swoje urządzenie
11. Odczytać wcześniej zapisany opis animacji
12. Zapisać animację na swoje urządzenie

13. Podzielić się wynikiem na Facebooku :)
14. Przygotowanie zadania, bez edytora tylko z pliku konfiguracyjnego
15. Logować się na serwer

Administrator powinien móc:

1. Płynnie przełączać się pomiędzy typem prostym double, a klasą mpf implementującą liczby zmien-
noprzecinkowe dowolnej dokładności
2. Wybierać metodę zrównoleglania
 - piksele
 - linie
 - klatki
 - części animacji
3. Wybierać liczbę jednostek wykonawczych

Aplikacja powinna:

1. Zarządzać zadaniami, kolejkować je
2. Sekwencyjnie wykonywać zadania dla wielu użytkowników (z priorytetowaniem)

2 Prototyp

2.1 Opis ogólny

1. Przynajmniej 2 liczące slave'y
2. Może być jednoużytkownikowy system – zlecamy zadanie i trzymamy na serwerze, żeby można było wrócić do niego po jakimś czasie.
3. Wynikiem może być wykonanie fraktala (1 klatki) lub film – dowolność.
4. Zlecenie zadania z np. wczytywaną konfiguracją z pliku, przekazanie do serwera, zakończenie zadania i zwrócenie wyniku użytkownikowi.

2.2 Opis szczegółowy

1. Klaster złożony z co najmniej dwóch maszyn, na każdej wykorzystywane wszystkie wątki procesora
2. Interfejs webowy umożliwiający zapamiętanie kolejki zadań
3. Możliwość pobrania wygenerowanego filmu (ew. obrazka)
4. Możliwość wpisania dowolnej funkcji zespolonej
5. Określenie czasu trwania animacji
6. Generowanie animacji na podstawie klatek kluczowych
7. Każda klatka kluczowa opisana współrzędnymi dwóch rogów obrazka
8. Wybór rozdzielczości animacji
9. Wybór między odcieniami szarości a kolorowym obrazkiem
10. Opis sceny wczytany z pliku tekstowego przygotowanego w edytorze tekstu
11. Interfejs webowy bez podziału na użytkowników uruchomiony na komputerze z aplikacją master
12. Kolejka może być obsługiwana przez aplikację odpowiadającą za interfejs webowy

3 System w wersji finalnej

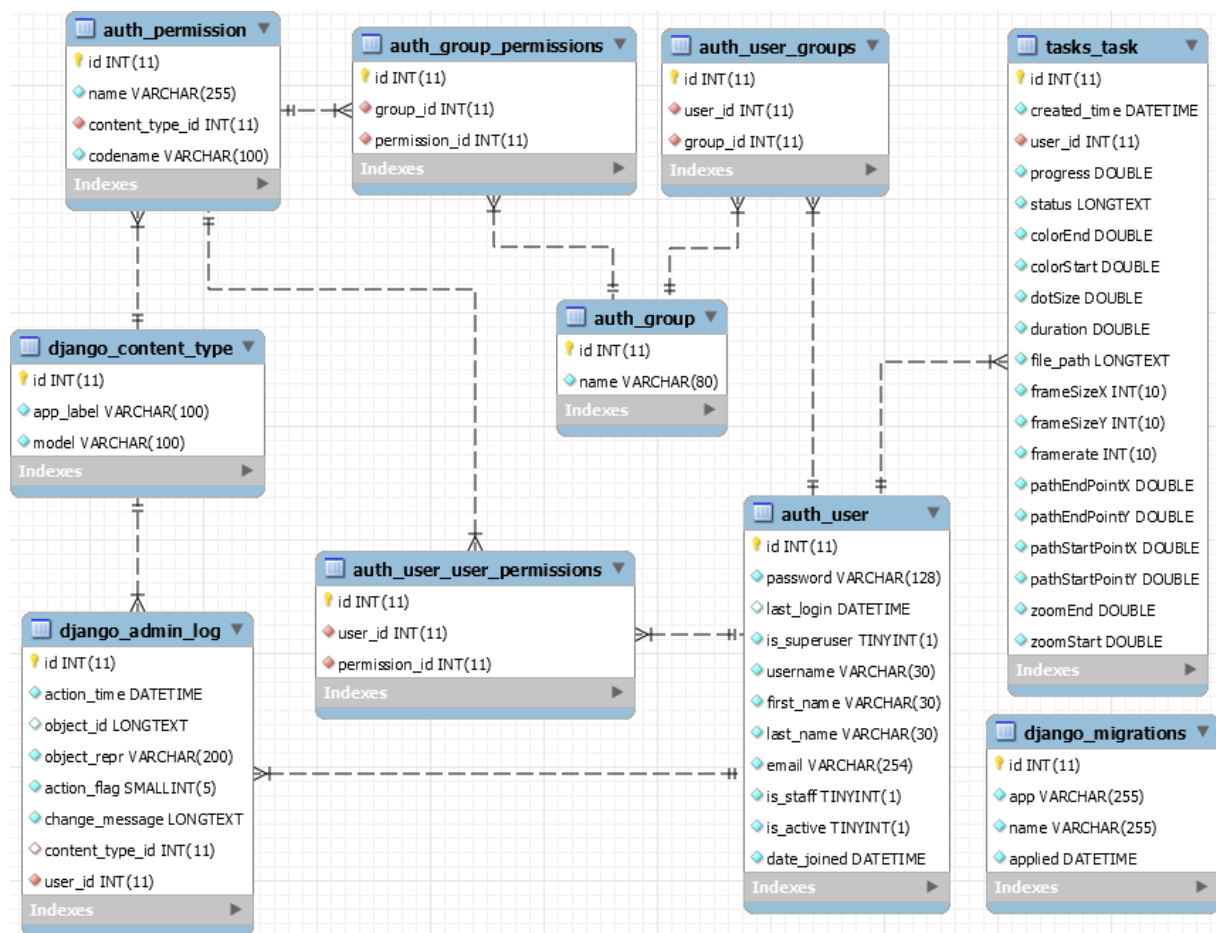
Aplikacja spełnia wymagania. Posiada czytelny interfejs użytkownika umożliwiający wprowadzanie nowych danych celem wykonania obliczeń. Wynikiem tychże jest animacja, której elementem jest fraktal.

Na działający serwis składa się kilka elementów. Są to:

- Serwer bazodanowy – MySQL Server
- Klaster MPI generujący animacje
- Serwer internetowy

3.1 Baza danych

Baza danych służy do przesyłania zadań pomiędzy aplikacją w Django, a aplikacją rozproszoną oraz do zarządzania użytkownikami systemu. Umieszczona jest na serwerze MySQL. Poniżej widoczny jest jej model fizyczny.



Rysunek 1: Model bazy danych

Kluczową dla zapewnienia funkcjonalności serwisu jest tabela tasks_task. Zawiera ona informację o zadaniach (zarówno parametry i status wykonania, jak i informację o dacie dodania i autorze).

Tabela auth_user zawiera użytkowników. Te i pozostałe tabele obsługuje serwer Django, między innymi na nich opierając swoje działanie.

3.2 MPI

Program co ustalony kwant czasu odpytuje bazę danych, sprawdzając czy są nowe zadania do wykonania. Jeśli znajdzie takowe, zaczyna je liczyć, zaprzęgając do pracy węzły MPI.

Główna funkcja programu jest dość prosta. W zależności od roli, jaką ma pełnić program, uruchamiany jest master lub slave.

```
1 #include "FractalCalc.h"
2 #include "Display.h"
3 #include "Utils.h"
4 #include "Communicator.h"
5 #include "Master.h"
6 #include "Slave.h"
7
8 double zoom = 1.01;
9 extern bool noDisplay;
10
11 int main(int argc, char** argv)
12 {
13     // compareArguments(argc, argv);
14
15     // MPI initialization
16     MPI_Init(&argc, &argv);
17     registerMPIDataTypes();
18     // Get the number of processes
19     int world_size;
20     MPI_Comm_size(MPLCOMM_WORLD, &world_size);
21     // Get the rank of the process
22     int rank;
23     MPI_Comm_rank(MPLCOMM_WORLD, &rank);
24     if(rank == 0)
25     {
26         Master m;
27         system("echo Master: $(hostname)");
28         m.work(argc, argv);
29     }
30     else
31     {
32         Slave s;
33         s.work(argc, argv);
34     }
35
36     // Finalize the MPI environment.
37     MPI_Finalize();
38     return 0;
39 }
```

3.2.1 Master

Master odpowiada za sprawdzanie bazy danych.

```
1 MysqlComm com(HOST, USER, PASS, DB);
2 try
3 {
4     com.Init();
5     com.Connect();
6     do
```

```

7 {
8     sleep(1);
9     com.Select("tasks_task", "*");
10    cout<<" Bool: " <<com.AnythingToDo()<<endl;
11 }
12 while( ! com.AnythingToDo() );
13 } catch(const char* e){
14     printf("%s\n", e);
15     fflush(stdout);
16 }

```

Jeśli znajdzie zadanie, pobiera jego parametry i rozsyła zadanie slave'om. Po odebraniu zadania aktualizowany jest jego status, dzięki czemu na stronie internetowej widać pasek postępu – o którym w dalszej części dokumentacji.

```

1 Task task = com.GetTask();
2 Scene s = com.GetScene();
3 com.TaskStart();
4 com.PrintRow();
5
6 vector<Order> orders(ordersCount);
7 int orderLength = generateOrders(orders, s, s.frameSize.x * s.frameSize.y / atoi(argv[1])
8     );
9
10 int ordersPendingCount = ordersCount;
11 system("echo start_tasks $(date +%X) $(( $(date +%s%N)/1000000) >> time.log");
12 for(int i = 1; i < world_size; ++i)
13 {
14     sendOrder(orders[ordersCount - ordersPendingCount], i, WORKTAG);
15     ordersPendingCount--;
16 }
17 map<int, vector<double>> results;
18 while(ordersPendingCount > 0)
19 {
20     receiveResult(results, MPLANY_SOURCE);
21     sendOrder(orders[orders.size() - ordersPendingCount], status.MPLSOURCE, WORKTAG);
22     ordersPendingCount--;
23     int progress = 100 - (ordersPendingCount*100) / ordersCount;
24     com.TaskUpdateProgress(progress);
25     cout << progress << endl;
26 }

```

Slave'y pobierają zadania – pojedyncze klatki do wygenerowania. Wygenerowane obrazy odbierane są od slave'ów:

```

1 for(; orderID < ordersCount; ++orderID)
2 {
3     if( orderID > 0 && (orderID % ordersPerFrame == 0) )
4     {
5         stringstream ss;
6         ss<<"images/"<<(orderID / ordersPerFrame)<<" .bmp";
7         image.save_image(ss.str().c_str());
8         cout << "IMG " <<ss.str()<<endl;
9     }
10    for(int i = 0; i < results[orderID].size() / 3; ++i)
11    {

```



```

12     r = results[orderID][3*i+0];
13     g = results[orderID][3*i+1];
14     b = results[orderID][3*i+2];
15
16     image.set_pixel(x++, y, r, g, b);
17     int tempX = x;
18     x = tempX % s.frameSize.x;
19     y = (y + (tempX / s.frameSize.x)) % s.frameSize.y;
20 }
21 }

```

Kiedy wszystkie zadania slave'ów zostaną wykonane, master łączy klatki w film za pomocą programu ffmpeg:

```

1 stringstream ss;
2 ss << "ffmpeg -framerate 30 -y -i images/%d.bmp -c:v libx264 -r 30 -pix_fmt yuv420p " <<
   task.id << ".mp4";
3 system(ss.str().c_str());

```

3.2.2 Slave

Slave jest odpowiedzialny za generowanie fraktala.

```

1 void Slave::work(int &argc, char** &argv)
2 {
3     int64_t size = 0;
4     int tag = 0;
5     while(doWork)
6     {
7         waitForOrder(order);
8         doWork = order.doWork;
9         size = 0;
10
11         if(true == order.doWork)
12         {
13             size = executeOrder(order, resultArray);
14             if(size > 0)
15             {
16                 int64_t id = order.orderID;
17                 sendResult(id, resultArray, size);
18             }
19             else
20                 doWork = false;
21         }
22         else
23         }
24 }

```

Wykonywanie zadania na poziomie slave'a odbywa się w metodzie executeOrder, a jego wynik dostępny w wektorze resultArray (polu klasy Slave).

```

1 int64_t Slave::executeOrder(Order &order, vector<double> &resultArray)
2 {
3     int64_t size = 0;

```

```

4  if (order.count > 0)
5  {
6      resultArray.resize (order.count);
7      size = FractalCalc::calcMandelbrotPart (resultArray.data(), order);
8      vector<double> colorArray (3*size);
9      int x = order.beginX;
10     int y = order.beginY;
11     for (int i = 0; i < order.count; ++i)
12     {
13         double color = resultArray[i];
14         double r, g, b;
15         r = 0.5*(sin(2*M_PI*color - M_PI/2 + M_PI /3) + 1) * 255.0;
16         g = 0.5*(sin(2*M_PI*color - M_PI/2) + 1) * 255.0;
17         b = 0.5*(sin(2*M_PI*color - M_PI/2 - M_PI / 3) + 1) * 255.0;
18         colorArray[3*i + 0] = r;
19         colorArray[3*i + 1] = g;
20         colorArray[3*i + 2] = b;
21         int tempX = ++x;
22         x = tempX % order.pictureWidth;
23         y = (y + (tempX / order.pictureWidth)) % order.pictureHeight;
24
25     }
26     resultArray.swap (colorArray);
27     size = resultArray.size();
28 }
29 return size;
30 }

```

3.2.3 Mandelbrot – generowanie fraktala

Slave wywołuje metodę liczącą fraktal – calcMandelbrotPart – z klasy FractalCalc.

```

1  class FractalCalc
2  {
3  public:
4      typedef void (* FractalFnPtrD) (complex<double>&, complex<double>&);
5      static const int convergenceSteps;
6      static const double divergenceLimitD;
7      //double
8      static void mandelbrotFractal (complex<double> &z, complex<double> &c);
9      static double getConvergence (double z_real, double z_imag, double c_real, double c_imag
10      , FractalFnPtrD fn);
11      static int calcMandelbrotPart (double* mandelbrot, Order &order);
12  };
13
14  void FractalCalc::mandelbrotFractal (complex<double> &z, complex<double> &c)
15  {
16      z = z * z + c;
17  }
18
19  double FractalCalc::getConvergence (double z_real, double z_imag, double c_real, double
20      c_imag, FractalFnPtrD fn)
21  {
22      complex<double> z (z_real, z_imag);
23      complex<double> c (c_real, c_imag); // c_real => c(e) real xD
24
25      // Sprawdzanie zbieznosci

```

```

24  int j = 0;
25  for (; j < convergenceSteps && abs(z) < divergenceLimitD; j++)
26  {
27      fn(z, c);
28  }
29  double log_zn, nu, result;
30  if (j < convergenceSteps)
31  {
32      log_zn = log(abs(z));
33      nu = log(log_zn / log(2)) / log(2);
34      result = (j + 1 - nu) / convergenceSteps;
35  }
36  return result;
37 }
38
39 int FractalCalc::calcMandelbrotPart(double* mandelbrot, Order &order)
40 {
41     int iX = order.beginX;
42     int iY = order.beginY;
43     int i = 0;
44     for (; i < order.count; ++i, ++iX)
45     {
46         if(iX >= order.pictureWidth)
47         {
48             iX = 0;
49             ++iY;
50         }
51         double dX = (order.fractalX - order.pictureWidth/2 * order.dotSize) + order.dotSize *
iX;
52         double dY = (order.fractalY - order.pictureHeight/2 * order.dotSize) + order.dotSize *
iY;
53         mandelbrot[i] = getConvergence(0, 0, dX, dY, mandelbrotFractal);
54     }
55     return i;
56 }

```

3.3 Django

Aplikacja napisana z użyciem frameworku Django jest kolejnym ważnym elementem systemu. To ona odpowiada za komunikację użytkownik – baza danych.

Widoki dostępne w interfejsie użytkownika, to strona główna, logowanie, rejestracja, dodawanie nowego fraktala i przeglądanie już dodanych w postaci wygenerowanych animacji. Poniżej umieszczono przykładowy szablon strony głównej i strony Moje animacje:

```

1  {% load staticfiles %}
2  <!DOCTYPE html>
3  <head>
4  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5  <title>{{ site.name }}</title>
6  <meta name="keywords" content="" />
7  <meta name="description" content="" />
8  <link href="http://fonts.googleapis.com/css?family=Varela" rel="stylesheet" />
9  <link href="{% static 'default.css' %}" rel="stylesheet" />
10 <link href="{% static 'fonts.css' %}" rel="stylesheet" />
11 <!--
12 <script src="//code.jquery.com/jquery-1.11.1.min.js"></script>

```

```

13 <script src="{% static 'js/test.js' %}"></script>
14 -->
15 <!--[if IE 6]><link href="{% static 'default_ie6.css' %}" rel="stylesheet" type="text/css
    " /><![endif]-->
16 </head>
17 <body>
18 <div id="wrapper">
19   <div id="header-wrapper">
20     <div id="header" class="container">
21       <div id="logo">
22         <h2><a href="{% url 'index' %}">FRACTAL<br />GENERATOR</a></h2>
23       </div>
24       {% block menu %}
25       {% endblock %}
26     </div>
27   </div>
28   <div id="extra" class="container">
29     {% block reg_log %}
30     {% if user.is_anonymous %}
31       {% block main %}
32       {% endblock %}
33     {% endif %}
34     {% endblock %}
35     {% if user.is_authenticated %}
36     {% block content %}
37     {% endblock %}
38     {% endif %}
39   </div>
40 </div>
41 <div id="copyright" class="container">
42   <p>&copy; Untitled. All rights reserved. | Photos by <a href="http://fotograph.com/">
    Fotogrph</a> | Design by <a href="http://templated.co" rel="nofollow">TEMPLATED</a>
    </p>
43 </div>
44 </body>
45 </html>

```

```

1 {% extends "tasks/index.html" %}
2 {% load staticfiles %}
3 {% block menu %}
4   <div id="menu">
5     <ul>
6       <li><a href="{% url 'index' %}" accesskey="1" title="">Strona glowna</a></li>
7         {% if user.is_authenticated %}
8         <li><a href="{% url 'generate' %}" accesskey="2" title="">Generuj fraktal</a></li>
9         <li><a href="{% url 'fractals' %}" accesskey="3" title="">Moje fraktale</a></li>
10        <li class="current_page_item"><a href="{% url 'animations' %}" accesskey="4"
    " title="">Moje animacje</a></li>
11        <li><a href="{% url 'auth_logout' %}" accesskey="5" title="">Wyloguj</a></li>
12      </ul>
13      {% else %}
14      <li><a href="{% url 'auth_login' %}" accesskey="2" title="">
    Zaloguj</a></li>
15      {% endif %}

```

[illegible]

Formularz rejestracji generuje Django.

```

1 {% extends "tasks/tasks.html" %}
2 {% load i18n %}
3
4 {% block reg_log %}
5     <form method="post" action=".">
6         {% csrf_token %}
7         {{ form.as_p }}
8         <label>&nbsp;</label><input type="submit" value="{% trans 'Submit' %}" />
9     </form>
10 {% endblock %}

```

Poniżej widoczne są przykładowe kody w Pythonie.

Widoki w pliku `views.py`:

```

1 from django.shortcuts import redirect, render
2 from django.contrib.auth.decorators import login_required
3 from .forms import AddTaskForm
4 from .models import Task
5
6 def get_dotsize(value):
7     return 4.0 / value
8
9 def index(request):
10     if request.user.is_authenticated():
11         tasks = Task.objects.filter(user=request.user).order_by('-created_time')
12         return render(request, 'tasks/tasks.html', {'tasks': tasks})
13     else:
14         return render(request, 'tasks/tasks.html', {})

```

```

15
16 @login_required(login_url='/')
17 def add(request):
18     if request.method == "POST":
19         form = AddTaskForm(request.POST)
20         if form.is_valid():
21             post = form.save(commit=False)
22             post.user = request.user
23             post.dotSize = get_dotsize(post.frameSizeX)
24             post.save()
25             return redirect('/', pk=post.pk)
26         else:
27             form = AddTaskForm()
28         return render(request, 'tasks/addTask.html', {'form': form})
29
30 @login_required(login_url='/')
31 def delete(request, task_id):
32     Task.objects.filter(user_id=request.user.id, id=task_id).delete()
33     return redirect('/')
34
35 @login_required(login_url='/')
36 def fractals(request):
37     return render(request, 'tasks/fractals.html', {})
38
39 @login_required(login_url='/')
40 def animations(request):
41     if request.user.is_authenticated():
42         tasks = Task.objects.filter(user=request.user, status='done').order_by('-
43         created_time')
44         return render(request, 'tasks/animations.html', {'videos': tasks})
45     else:
46         return render(request, 'tasks/animations.html', {})

```

Plik z ustawieniami wygląda następująco:

```

1 import os
2
3 # Build paths inside the project like this: os.path.join(BASE_DIR, ...)
4 BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
5
6 # SECURITY WARNING: keep the secret key used in production secret!
7 SECRET_KEY = '6=saxft6ij11khby3r@((u*frkyo9e5c!49_!=sd0k64f1qp&'
8
9 # SECURITY WARNING: don't run with debug turned on in production!
10 DEBUG = True
11
12 ALLOWED_HOSTS = []
13
14 # Application definition
15
16 INSTALLED_APPS = [
17     'tasks.apps.TasksConfig',
18     'django.contrib.admin',
19     'django.contrib.auth',
20     'django.contrib.contenttypes',
21     'django.contrib.sessions',
22     'django.contrib.messages',
23     'django.contrib.staticfiles',

```

```

24 ]
25
26 MIDDLEWARE_CLASSES = [
27     'django.middleware.security.SecurityMiddleware',
28     'django.contrib.sessions.middleware.SessionMiddleware',
29     'django.middleware.common.CommonMiddleware',
30     'django.middleware.csrf.CsrfViewMiddleware',
31     'django.contrib.auth.middleware.AuthenticationMiddleware',
32     'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
33     'django.contrib.messages.middleware.MessageMiddleware',
34     'django.middleware.clickjacking.XFrameOptionsMiddleware',
35 ]
36
37 ROOT_URLCONF = 'mandelbrot.urls'
38
39 TEMPLATES = [
40     {
41         'BACKEND': 'django.template.backends.django.DjangoTemplates',
42         'DIRS': [],
43         'APP_DIRS': True,
44         'OPTIONS': {
45             'context_processors': [
46                 'django.template.context_processors.debug',
47                 'django.template.context_processors.request',
48                 'django.contrib.auth.context_processors.auth',
49                 'django.contrib.messages.context_processors.messages',
50             ],
51         },
52     },
53 ]
54
55 WSGIAPPLICATION = 'mandelbrot.wsgi.application'
56
57 # Database
58 # https://docs.djangoproject.com/en/1.9/ref/settings/#databases
59
60 DATABASES = {
61     'default': {
62         'ENGINE': 'django.db.backends.mysql',
63         'NAME': 'mandel',
64         'HOST': 'adres.bazy.com',
65         'PORT': '3306',
66         'USER': 'mandelbrot',
67         'PASSWORD': 'haslodobazy',
68     }
69 }
70
71 # Password validation
72 # https://docs.djangoproject.com/en/1.9/ref/settings/#auth-password-validators
73
74 AUTHPASSWORD_VALIDATORS = [
75     {
76         'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
77     },
78     {
79         'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
80     },
81     {

```

```

82         'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
83     },
84     {
85         'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
86     },
87 ]
88
89 # Internationalization
90 # https://docs.djangoproject.com/en/1.9/topics/i18n/
91
92 LANGUAGE_CODE = 'pl-pl'
93
94 TIME_ZONE = 'Europe/Warsaw'
95
96 USE_I18N = True
97
98 USE_L10N = True
99
100 USE_TZ = True
101
102 # Static files (CSS, JavaScript, Images)
103 # https://docs.djangoproject.com/en/1.9/howto/static-files/
104
105 STATIC_ROOT = "/root/mandelbrot/static/"
106 STATIC_URL = '/static/'
107 STATICFILES_DIRS = (
108     os.path.join("videos", "/home/dave/"),
109 )
110
111 REGISTRATION_OPEN = True # If True, users can register
112 ACCOUNT_ACTIVATION_DAYS = 7 # One-week activation window; you may, of course, use a
    different value.
113 REGISTRATION_AUTO_LOGIN = False # If True, the user will be automatically logged in.
114 LOGIN_REDIRECT_URL = '/' # The page you want users to arrive at after they successful
    log in
115 LOGIN_URL = '/accounts/login/' # The page users are directed to if they are not logged
    in,
116 # and are trying to access pages requiring authentication
117
118 EMAIL_USE_TLS = True
119 EMAIL_HOST = 'smtp.gmail.com'
120 EMAIL_PORT = 587
121 EMAIL_HOST_USER = 'fractalgenerator@gmail.com'
122 EMAIL_HOST_PASSWORD = 'haslodokontapocztowego'

```

Obsługa formularza:

```

1 from django import forms
2 from .models import Task
3
4 class AddTaskForm(forms.ModelForm):
5     framerates = (
6         ('24', '24'),
7         ('30', '30'),
8         ('60', '60'),
9     )
10
11     frameSizeX = forms.IntegerField(required=True, label="Szerokosc")

```



```

12     frameSizeY = forms.IntegerField(required=True, label="Dlugosc")
13     framerate = forms.ChoiceField(choices=framerates, required=True, label="Liczba klatek /sekunde")
14     duration = forms.FloatField(required=True, label="Czas Trwania", help_text="(s)")
15     pathStartPointX = forms.FloatField(required=True, label="Punkt Startowy X")
16     pathStartPointY = forms.FloatField(required=True, label="Punkt Startowy Y")
17     pathEndPointX = forms.FloatField(required=True, label="Punkt Koncowy X")
18     pathEndPointY = forms.FloatField(required=True, label="Punkt Koncowy Y")
19     zoomStart = forms.FloatField(required=True, min_value=1, label="Powiekszenie startowe")
20     zoomEnd = forms.FloatField(required=True, min_value=1, label="Powiekszenie koncowe")
21     # colorStart = forms.FloatField()
22     # colorEnd = forms.FloatField()
23
24     class Meta:
25         model = Task
26         exclude = [ 'dotSize', 'progress', 'user', 'file_path', 'status', 'created_time',
27                     'colorStart', 'colorEnd' ]

```

Zadanie:

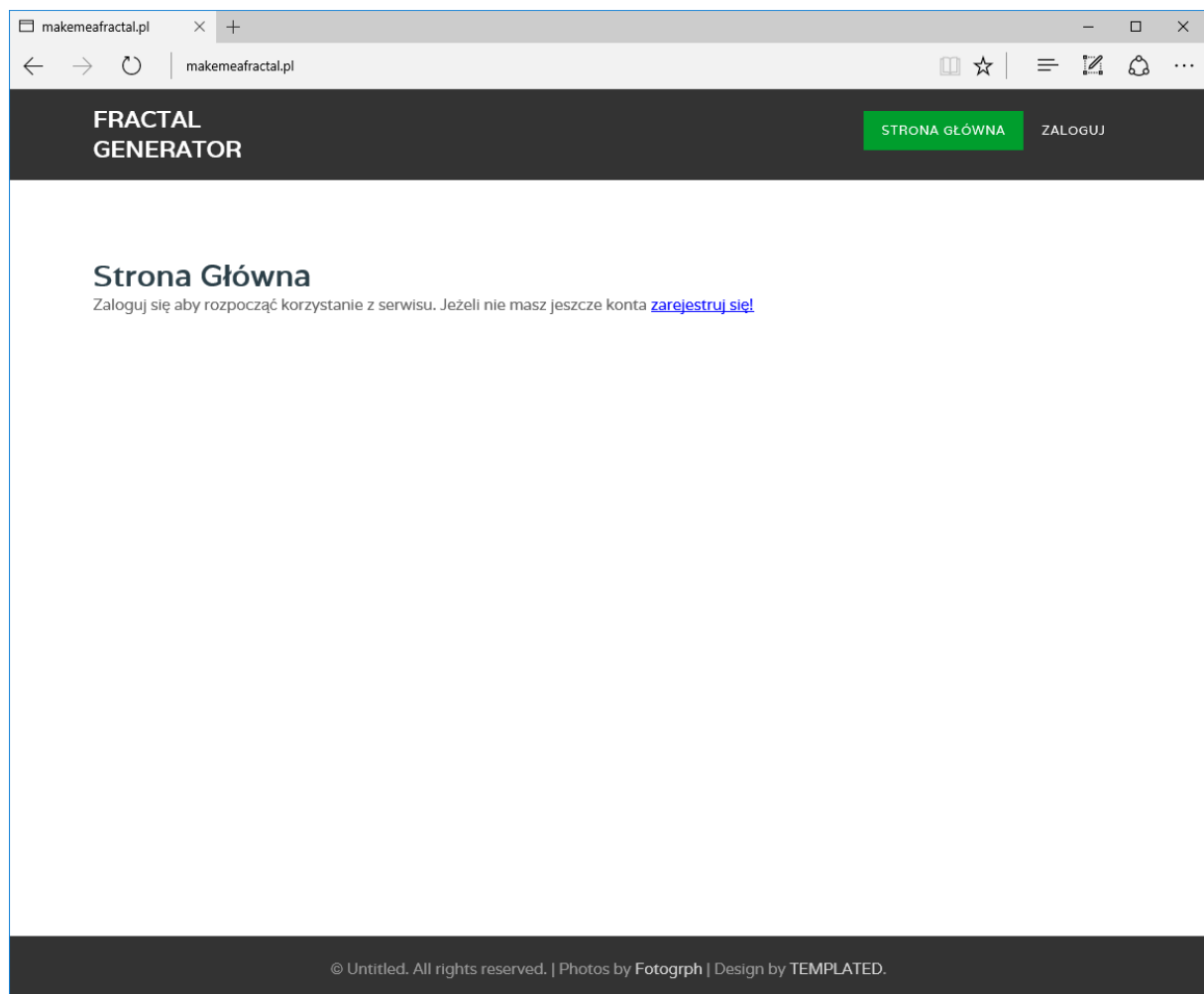
```

1 from django.db import models
2 from django.contrib.auth.models import User
3 from django.utils import timezone
4
5 class Task(models.Model):
6     user = models.ForeignKey(User, on_delete=models.CASCADE, )
7     created_time = models.DateTimeField(default=timezone.now, )
8     file_path = models.TextField(default="", )
9     status = models.TextField(default="pending", )
10    progress = models.FloatField(default="0", )
11    duration = models.FloatField(default="0", )
12    framerate = models.PositiveIntegerField(default="0")
13    frameSizeX = models.PositiveIntegerField(default="0")
14    frameSizeY = models.PositiveIntegerField(default="0")
15    dotSize = models.FloatField(default="0")
16    pathStartPointX = models.FloatField(default="0")
17    pathStartPointY = models.FloatField(default="0")
18    pathEndPointX = models.FloatField(default="0")
19    pathEndPointY = models.FloatField(default="0")
20    zoomStart = models.FloatField(default="0")
21    zoomEnd = models.FloatField(default="0")
22    colorStart = models.FloatField(default="0")
23    colorEnd = models.FloatField(default="0")
24
25    def get_dotsize(self):
26        return 4.0 / self.frameSizeX

```

3.4 Strona internetowa – interfejs użytkownika

Poniższe zrzuty ekranu prezentują działanie serwisu:



Rysunek 2: Strona główna

Na stronie głównej widoczna jest propozycja zalogowania się, lub rejestracji. Bez tego niemożliwe jest wysyłanie zadań do wykonania.

makemeafractal.pl

makemeafractal.pl/accounts/register

**FRACTAL
GENERATOR**

[STRONA GŁÓWNA](#) [ZALOGUJ](#)

Użytkownik: Wymagane. 30 znaków lub mniej. Tylko litery, cyfry i znaki @/./+/-/_.

Email: adres email

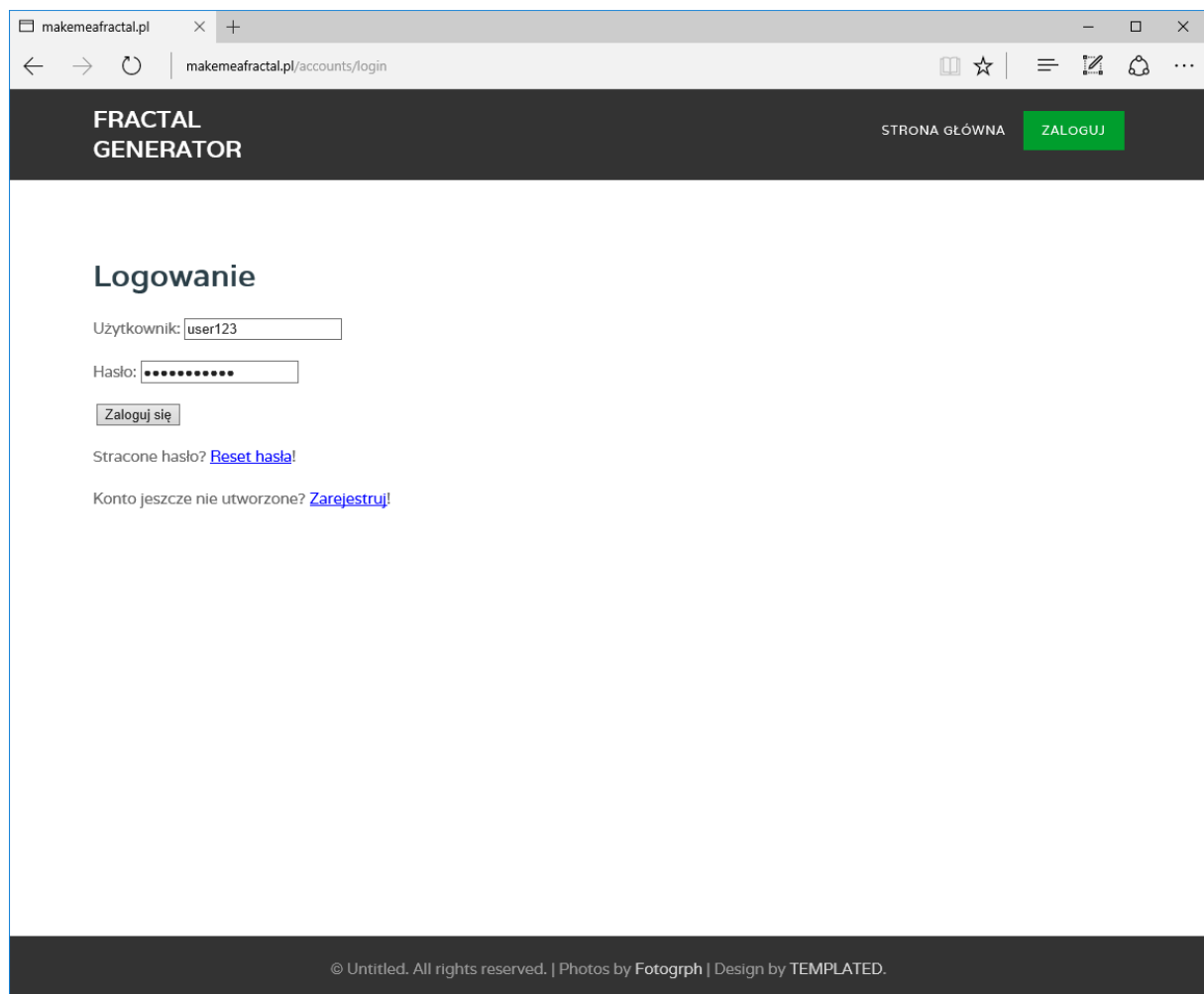
Hasło:

Potwierdzenie hasła: Wprowadź to samo hasło ponownie, dla weryfikacji.

© Untitled. All rights reserved. | Photos by Fotogrph | Design by TEMPLATED.

Rysunek 3: Rejestracja

Rejestracja wymaga podania nazwy użytkownika, adresu email i hasła. Na podany adres zostaje wysłany link aktywacyjny, po kliknięciu którego można korzystać z serwisu.



Rysunek 4: Logowanie

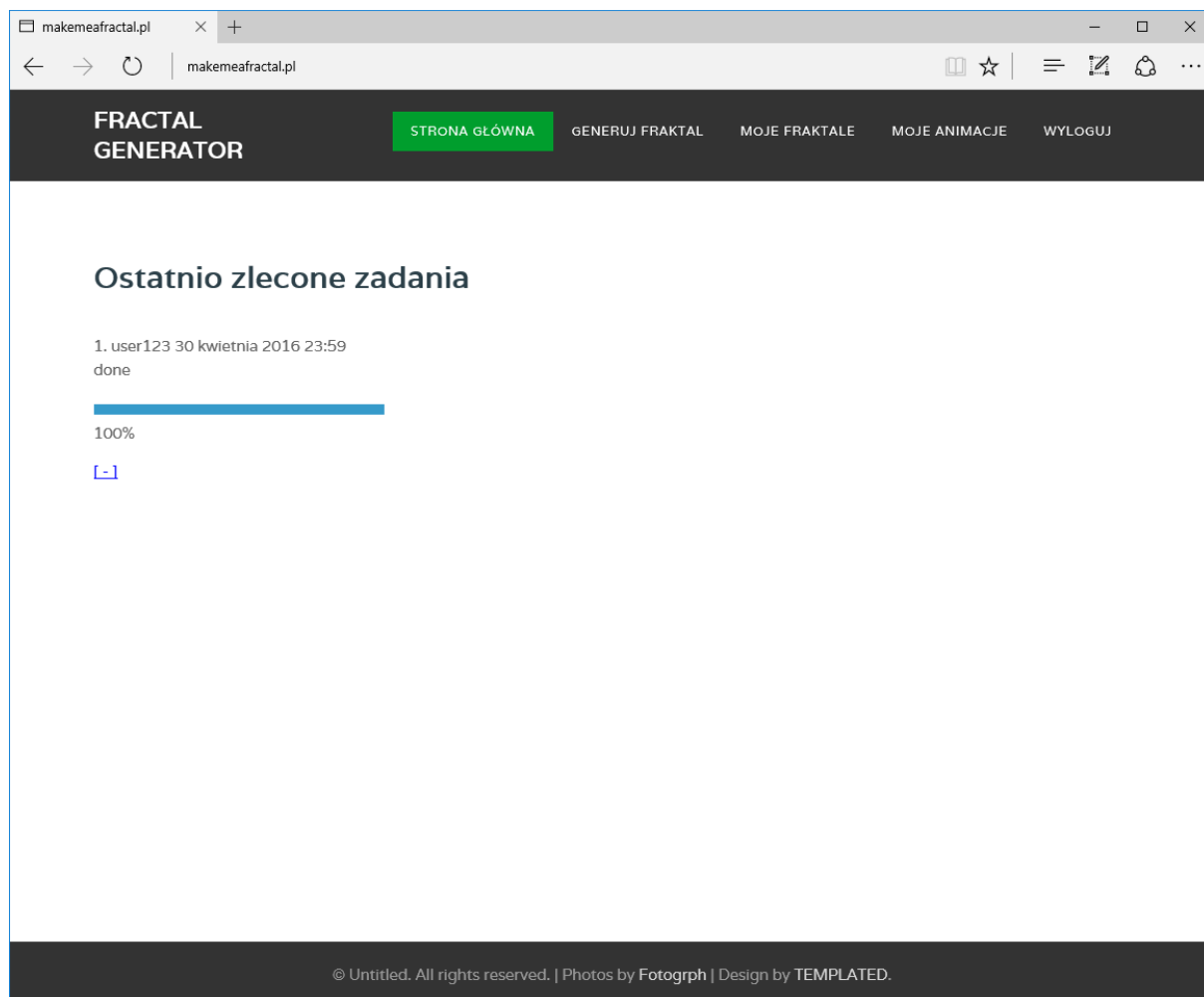
The screenshot shows a web browser window with the address bar displaying 'makemeafractal.pl/generate'. The website has a dark header with the title 'FRACTAL GENERATOR' and navigation links: 'STRONA GŁÓWNA', 'GENERUJ FRAKTAL' (highlighted in green), 'MOJE FRAKTALE', 'MOJE ANIMACJE', and 'WYLOGUJ'. The main content area is titled 'Generuj fraktal' and contains a form with the following fields:

- Czas Trwania: (s)
- Liczba klatek/sekunde:
- Szerokosc:
- Dlugosc:
- Punkt Startowy X:
- Punkt Startowy Y:
- Punkt Koncowy X:
- Punkt Koncowy Y:
- Powiekszenie startowe:
- Powiekszenie koncowe:

At the bottom of the form is a button labeled 'Dodaj'. The footer of the page contains the text: '© Untitled. All rights reserved. | Photos by Fotogrph | Design by TEMPLATED.'

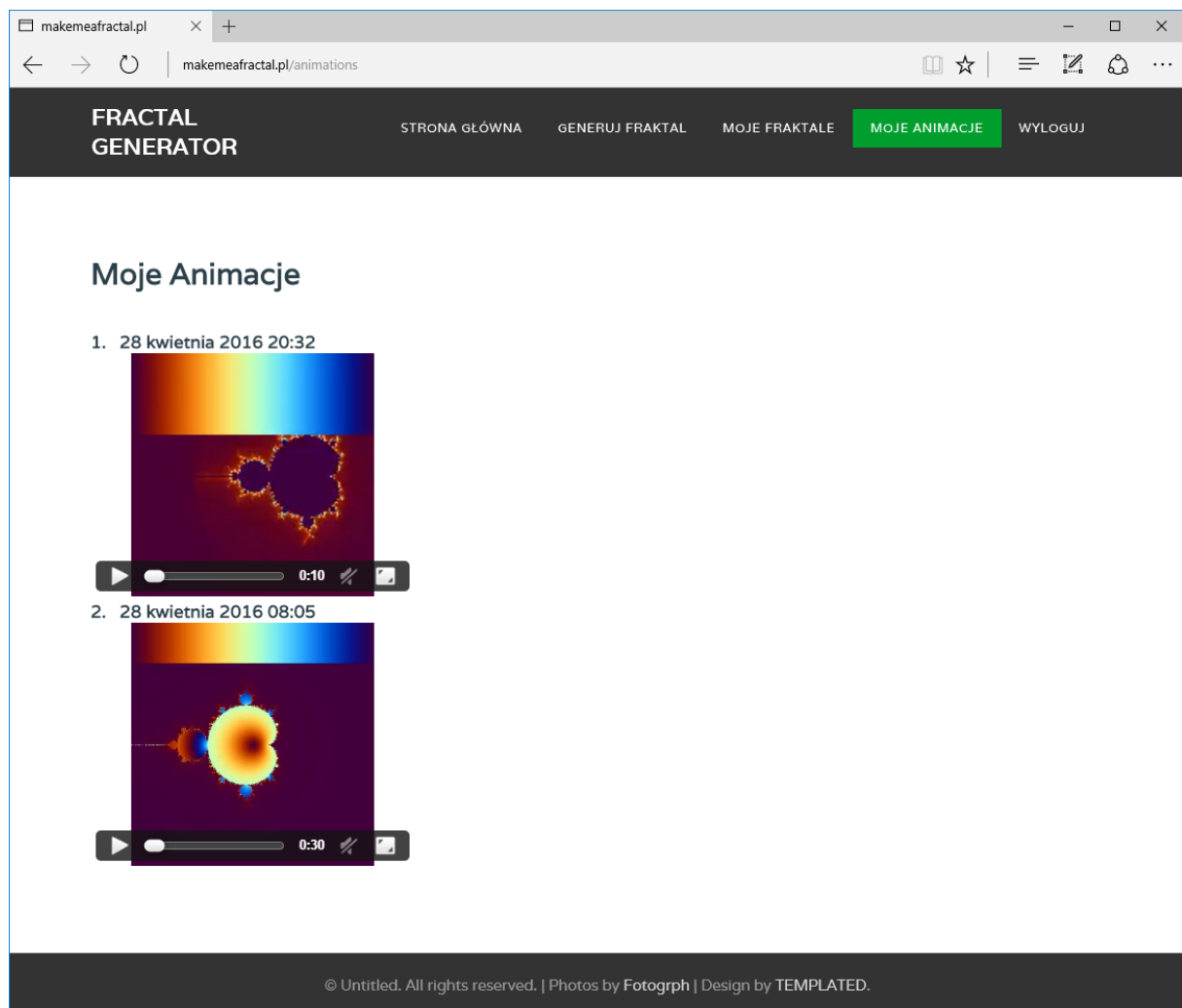
Rysunek 5: Generowanie fraktala

Na tej stronie istnieje możliwość dodania nowego zadania do wykonania. Podawane są parametry animacji - dotyczące zarówno pliku wideo, jak i samego fraktala i jego pozycji.



Rysunek 6: Kolejka zadań

Po zalogowaniu się na stronie głównej widoczne są zlecone zadania i ich status wykonania wraz z paskiem postępu.



Rysunek 7: Moje animacje

Na stronie Moje animacje widoczny jest podgląd wygenerowanych animacji.

4 Uruchamianie

W celu uruchomienia aplikacji potrzebne są następujące składniki:

- lib mysql
- libctdc++6
- serwer ssh
- authorizedkeys
- rsync
- openmpi

Program obsługujący MPI znajduje się w katalogu Cluster. Należy skompilować go poleceniem make. Model bazy danych (Doc/mandel.mvb) należy otworzyć za pomocą programu MySQL Workbench i na jego podstawie wypełnić bazę tabelami. Aplikacja internetowa znajduje się w katalogu WebApp.

5 Testy wydajności

Testy zostały wykonane przy użyciu trzech fizycznych maszyn. 2 z nich były wyposażone w 2 (intel i5), a jedna 4 (intel i7) rdzenie procesora. Zastosowana w nich technologia HyperTreading pozwala na jednoczesną pracę dwóch wątków na jednym rdzeniu, zatem można stwierdzić, że były dostępne odpowiednio 4 i 8 rdzeni. Maszyna na której uruchamiany był master pracowała pod kontrolą systemu operacyjnego Linux Mint, natomiast na pozostałych były zainstalowane maszyny wirtualne z systemem Linux Debian. Komputery były spięte do jednego switcha z portami FastEthernet. Dostępny sprzęt pozwolił przetestować następujące konfiguracje:

1. 1 maszyna fizyczna, 4 rdzenie
2. 1 maszyna fizyczna, 8 rdzeni
3. 2 maszyny fizyczne, 8 rdzeni (4 + 4)
4. 2 maszyny fizyczne, 12 rdzeni (4 + 8)
5. 3 maszyny fizyczne, 12 rdzeni (4 + 4 + 4)
6. 3 maszyny fizyczne, 16 rdzeni (4 + 4 + 8)

Na wszystkich maszynach zostało uruchomione zadanie wygenerowania 1-sekundowej animacji (30 klatek) w różnej rozdzielczości (400x400 – 2000x2000 px ze skokiem co 200 px). Drugi scenariusz testowy dotyczył podziału klatki na mniejsze podzadania. W tym przypadku rozdzielczość została ustalona na 2000 x 2000 px, natomiast klatka była dzielona na 1, 2, 4, 8, 16 podzadań (seed). Wszystkie pomiary zostały wykonane 10-krotnie, a następnie uśredniono wyniki. Widoczny na wykresach przypadek all dotyczył czasu wykonania całego zadania, wraz z generowaniem animacji z gotowych klatek, natomiast mpi dotyczy wyłącznie czasu obliczeń zrównoleglonych. Wykresy na których nie ma tego podziału pokazują czasy tylko obliczeń zrównoleglonych.



Rysunek 8: Porównanie czasu dla różnych konfiguracji

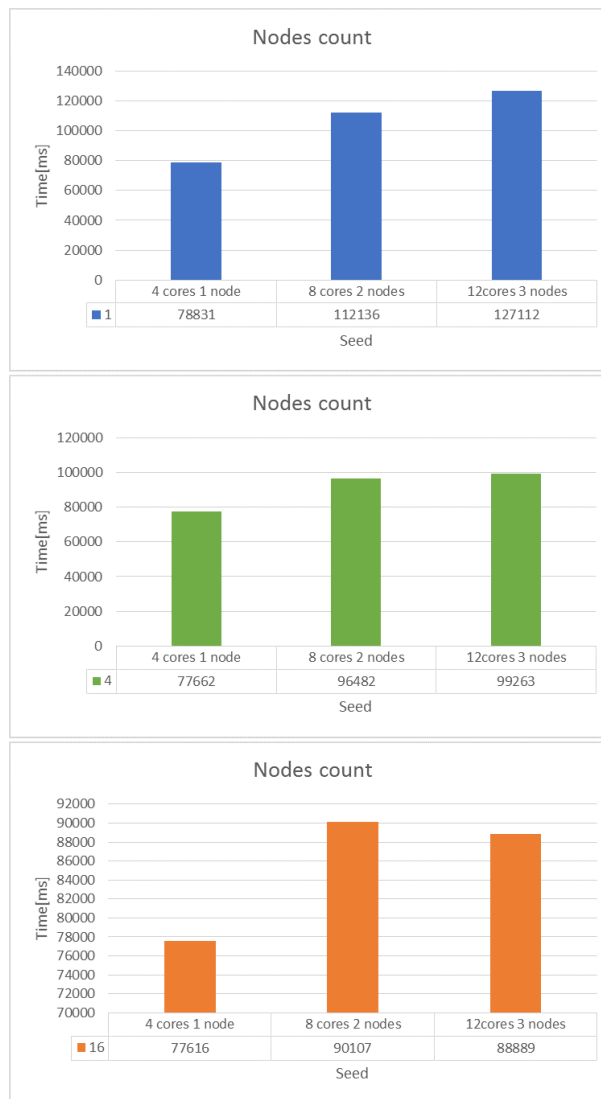
Z powyższych wykresów widać, że jeśli wszystkie rdzenie łączące znajdują się na jednej maszynie to podział zadania na różne fragmenty nie ma większego wpływu na czas obliczeń. Jeżeli jednak zwiększymy ilość maszyn liczących, to podział zadania na mniejsze pozwala zaobserwować spadek czasu obliczeń. Różnica ta zwiększa się wraz z dokładaniem do klastra kolejnych maszyn. Również liczba przydzielonych rdzeni procesora ma znaczący wpływ na szybkość omawianej zmiany.



Rysunek 9: Porównanie czasu dla różnych konfiguracji

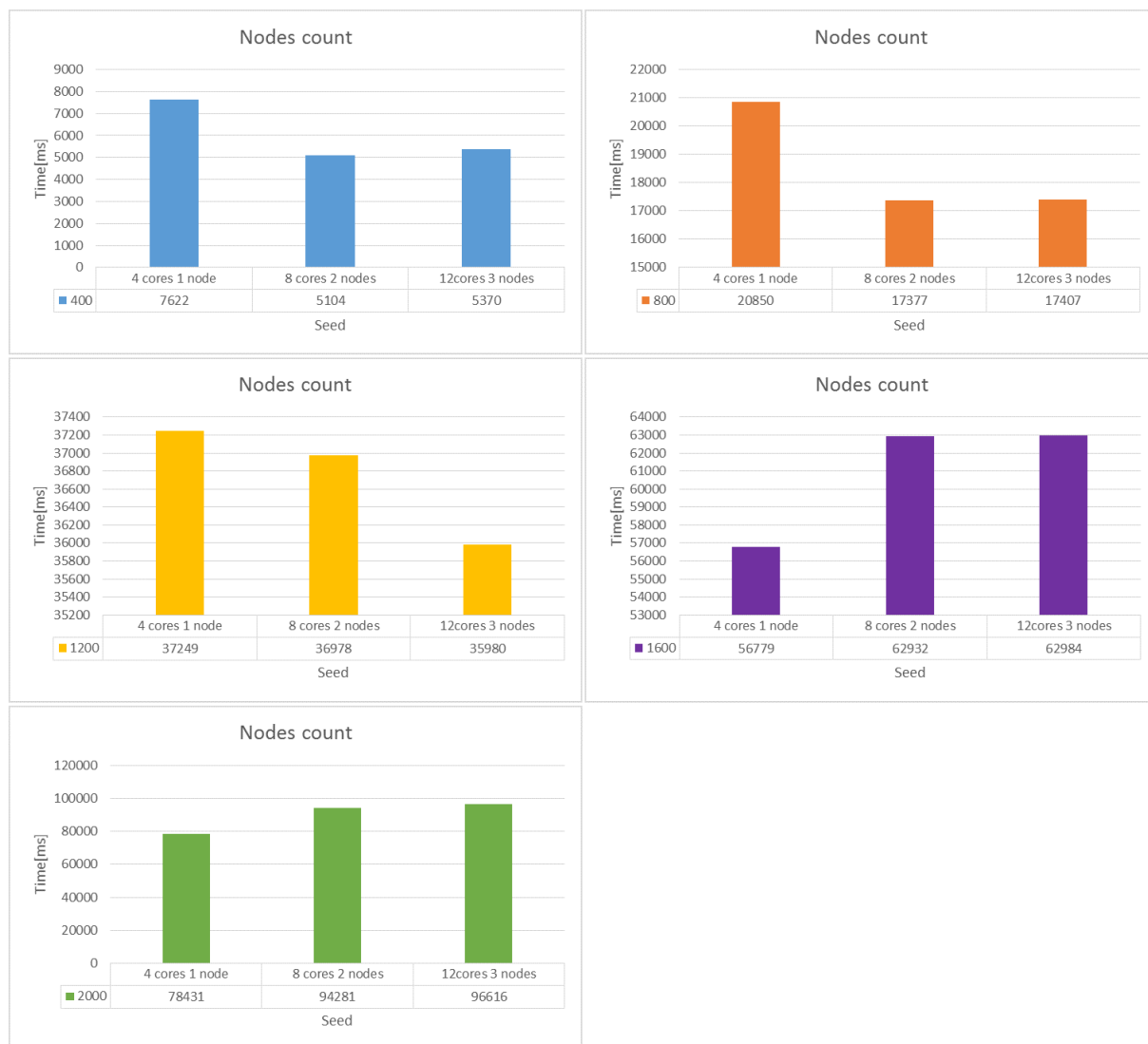
Porównanie czasu obliczeń w zależności od rozmiaru klatki obrazu ukazuje wyniki zgodne z oczekiwaniami. Ilość maszyn liczących i rdzeni procesora nią ma wpływu na charakter zmiany czasu obliczeń. We wszystkich przetestowanych przypadkach jest to zależność wielomianowa, zbliżona do funkcji kwadratowej. Ma to uzasadnienie w zmianie rozmiarze zadania – dwukrotne zwiększenie boku klatki powoduje czterokrotne zwiększenie rozmiaru zadania.

Otrzymane wyniki są dość niespodziewane. Wydawać by się mogło, że dodatkowe węzły liczące powinny powodować redukcję czasu potrzebnego na obliczenia. Tymczasem, nasze pomiary nie wykazują tej własności. Najprawdopodobniej użyty do łączenia maszyn interfejs FastEthernet ma zbyt małą przepustowość przez co procesory nie są maksymalnie wykorzystane i występuje długi czas oczekiwania na komunikację pomiędzy masterem i slavem.



Rysunek 10: Porównanie czasu dla różnych liczb podzadań

Na wszystkich wykresach można zaobserwować wzrost czasu trwania wykonania zadania zależnie od ilości połączonych węzłów. Jest to spowodowane narzutem komunikacji pomiędzy węzłem master a wszystkimi węzłami slave. Patrząc jednak na wielkość ziarna zadania, czyli podziału jednej klatki na kolejno 1, 4 i 16 podzadań można zauważyć, że większe rozdrobnienie idzie w parze z przyspieszeniem czasu wykonania całości zadania, jest to lepiej widoczne im więcej węzłów połączymy ze sobą. Większa ziarnistość powoduje zmniejszenie zużycia łącza na przesłanie dużego fragmentu danych (przy ziarnie 1 przesyłana jest cała klatka, a przy 16 tylko 1/16). Przesyłanie trwa na tyle krótko żeby nie doszło do zbyt długiego blokowania węzła master podczas gdy inne węzły slave czekają na swoją kolej obsługi.



Rysunek 11: Porównanie czasu dla różnych rozmiarów klatek

Na wykresach czasu trwania wykonania zadania od wielkości klatki w pikselach możemy zauważyć dwie tendencje dla czasu wykonania. Pierwsza z nich to spadkowa, jeśli zwiększamy ilość węzłów i jest ona zauważalna dla rozmiarów klatek 400, 800, 1200. Druga z nich to tendencja wzrostowa jeśli zwiększamy ilość węzłów i widać ją dla rozmiaru klatek 1600 i 2000. Tendencja spadkowa przy małych klatkach jest spowodowana zwiększeniem mocy obliczeniowej klastra. Natomiast tendencja wzrostowa wynika ze zbyt długiego czasu przesyłania wyniku do węzła master. W tym czasie inne węzły mogą ukończyć zadanie i niestety muszą czekać dłużej na swoją kolej obsługi. Drugim, dość oczywistym wnioskiem, jest to że im większy rozmiar klatki tym więcej czasu potrzeba na wykonanie zadania.

6 Podsumowanie

Projekt ten ukazał zalety wynikające z rozpraszania obliczeń na wielu maszynach. Ponadto pokazał, jak używać technologii webowych (frameworku Django, języka Python).