Distributed Workflows with Flowy

EuroPython 2015

Sever Banesiu @severb
Overview

1. Distributed Workflows
2. Code + Demo
3. Workflow Engine
4. Execution Model
5. More Examples
6. Scaling
What is a distributed workflow?

**Hint**
A process composed of a mix of independent and interdependent units of work called *tasks*. 
Workflows are usually modeled with DAGs or ad-hoc code.

*Note*
Neither provide a satisfactory solution.
Flowy
A Workflow Modeling Library

It uses **single-threaded**-looking Python code and gradual concurrency inference.
An Example
An Ad-hoc Solution, using task queues

- find chapters
- embed subtitle
- target ads

task queue
An Ad-hoc Solution, using task queues

- Task queue
- embed subtitle
- target ads
- find chapters
- worker
- storage
An Ad-hoc Solution, using task queues

task queue

- extract thumbnail
- find chapters
- embed subtitle
- target ads

worker

storage
An **Ad-hoc Solution**, using task queues

decision

embed subtitle

target ads

find chapters

worker

storage

task queue
An Ad-hoc Solution, using task queues

- Extract thumbnail
- Embed subtitle
- Target ads
- Decision
- Worker
- Storage

Task queue
The Workflow Engine

* automatically schedule the corresponding decision type when an activity is finished

* ensure all decisions for the same workflow execution are sequential

* merge multiple queued decisions for the same workflow execution into one

* provide fault tolerance with timers

---

activity queue

activity worker

activity

decision worker

decision

decision queue

storage

API
The Workflow Engine

Not something new
def process_video(embed_subtitle, find_chapters, ...):
    def workflow(video_URL, subtitle_URL):
        new_URL = embed_subtitle(video_URL, subtitle_URL)
        webm_URL = encode_video(new_URL, 'webm')
        mpeg4_URL = encode_video(new_URL, 'mpeg4')
        ad_tags = target_ads(subtitle_URL)
        chapters = find_chapters(video_URL)
        thumbnails = [extract_thumbnail(video_URL, c) for c in chapters]
        return video_URL, webm_URL, mpeg4_URL, thumbnails, ad_tags

    return workflow
Execution Model

```python
def process_video(embed_subtitle, find_chapters, ...):
    def workflow(video_URL, subtitle_URL):
        new_URL = embed_subtitle(video_URL, subtitle_URL)
        webm_URL = encode_video(new_URL, 'webm')
        mpeg4_URL = encode_video(new_URL, 'mpeg4')
        ad_tags = target_ads(subtitle_URL)
        chapters = find_chapters(video_URL)
        thumbnails = [extract_thumbnail(video_URL, c) for c in chapters]
        return video_URL, webm_URL, mpeg4_URL, thumbnails, ad_tags
    return workflow
```
Execution Model

def process_video(embed_subtitle, find_chapters, ...):
    def workflow(video_URL, subtitle_URL):
        new_URL = embed_subtitle(video_URL, subtitle_URL)
        webm_URL = encode_video(new_URL, 'webm')
        mpeg4_URL = encode_video(new_URL, 'mpeg4')
        ad_tags = target_ads(subtitle_URL)
        chapters = find_chapters(video_URL)
        thumbnails = [extract_thumbnail(video_URL, c) for c in chapters]
        return video_URL, webm_URL, mpeg4_URL, thumbnails, ad_tags
    return workflow
def process_video(embed_subtitle, find_chapters, ...):
    def workflow(video_URL, subtitle_URL):
        new_URL = embed_subtitle(video_URL, subtitle_URL)
        webm_URL = encode_video(new_URL, 'webm')
        mpeg4_URL = encode_video(new_URL, 'mpeg4')
        ad_tags = target_ads(subtitle_URL)
        chapters = find_chapters(video_URL)
        thumbnails = [extract_thumbnail(video_URL, c) for c in chapters]
        return video_URL, webm_URL, mpeg4_URL, thumbnails, ad_tags
    return workflow
Execution Model

def process_video(embed_subtitle, find_chapters, ...):
    def workflow(video_URL, subtitle_URL):
        new_URL = embed_subtitle(video_URL, subtitle_URL)
        webm_URL = encode_video(new_URL, 'webm')
        mpeg4_URL = encode_video(new_URL, 'mpeg4')
        ad_tags = target_ads(subtitle_URL)
        chapters = find_chapters(video_URL)
        thumbnails = [extract_thumbnail(video_URL, c) for c in chapters]
        return video_URL, webm_URL, mpeg4_URL, thumbnails, ad_tags
    return workflow
Execution Model

```python
def process_video(embed_subtitle, find_chapters, ...):
    def workflow(video_URL, subtitle_URL):
        new_URL = embed_subtitle(video_URL, subtitle_URL)
        webm_URL = encode_video(new_URL, 'webm')
        mpeg4_URL = encode_video(new_URL, 'mpeg4')
        ad_tags = target_ads(subtitle_URL)
        chapters = find_chapters(video_URL)
        thumbnails = [extract_thumbnail(video_URL, c) for c in chapters]
        return video_URL, webm_URL, mpeg4_URL, thumbnails, ad_tags
    return workflow
```
Side Effects

The execution path must not change between invocations.

Use only pure functions inside the workflow code.
Use input data or dedicated activities for random values, current date, external reading, etc.
Avoid complex computations in the workflow code.
Using Task Results

def example(square):
    def workflow(a, b):
        a_squared = square(a)
        b_squared = square(b)
        if a_squared + b_squared > 100:
            return math.copysign(a_squared, a)
        return math.copysign(b_squared, b)
    return workflow
Using Task Results

def example(square):
    def workflow(a, b):
        a_squared = square(a)
        b_squared = square(b)
        if a_squared + b_squared > 100:
            return math.copysign(a_squared, a)
        return math.copysign(b_squared, b)
    return workflow
def process_video(embed_subtitle, find_chapters, ...):
    def workflow(video_URL, subtitle_URL):
        new_URL = embed_subtitle(video_URL, subtitle_URL)
        webm_URL = encode_video(new_URL, 'webm')
        mpeg4_URL = encode_video(new_URL, 'mpeg4')
        ad_tags = target_ads(subtitle_URL)
        chapters = find_chapters(video_URL)
        thumbnails = [extract_thumbnail(video_URL, c) for c in chapters]
        return video_URL, webm_URL, mpeg4_URL, thumbnails, ad_tags
    return workflow
Using Task Results

def example(sum, square):
    def workflow(a, b):
        a_squared = square(a)
        b_squared = square(b)
        if a_squared < 100:
            a_squared = sum(a_squared, 100)
        if b_squared > 100:
            b_squared = sum(b_squared, 100)
        return sum(a_squared, b_squared)
    return workflow
Subworkflows

```python
def subworkflow(sum, square):
    def workflow(n):
        n_squared = square(n)
        if n_squared < 100:
            n_squared = sum(n_squared, 100)
        return workflow

    return workflow

def example(sum, example_sub):
    def workflow(a, b):
        return sum(example_sub(a_squared), example_sub(b_squared))

    return workflow
```
def example(square):
    def workflow(a):
        try:
            a_squared = square(a)
        except:
            return 0
        else:
            return a_squared + 100
    return workflow
Error Handling

def example(square):
    def workflow(a):
        a_squared = square(a)
        try:
            return a_squared + 100
        except TaskError:
            return 0
    return workflow
def example(square):
    def workflow(a):
        a_squared = square(a)
        try:
            wait(a_squared)
        except TaskError:
            return 0
        else:
            return a_squared + 100
    return workflow
Error Handling

```python
def example(sum, square):
    def workflow(a, b):
        a_squared = square(a)
        b_squared = square(b)
        return sum(a_squared, b_squared)
    return workflow
```
Scaling

* only configuration changes (+ heartbeat callable)
* execution timers for fault tolerance
* a new error type, TimeoutError
* automatic retries on timeout
* heartbeats
* idempotent activities
* activities in other languages
* results and input data size restrictions
* each worker is single threaded/process (use process managers)
* use subworkflows if history gets too large
* can scale up and down with ease (overall progress is not lost)
Thank you,
Questions?

github.com/severb/flowy/