RFC: Batching pytest

Author: Dan Moran

`./pants test` currently runs a separate `pytest` process per `python_test` target. The community has flagged this as a major source of performance issues when running a large number of tests under Pants (see this issue). Unfortunately, in a monorepo setting it's not always safe to batch arbitrary tests together in the same `pytest` process. Pants needs new syntax so users can mark which tests are safe / unsafe to batch together.

Why not arbitrary batches?

Mutable global state

Consider a monorepo containing multiple Django projects, with a layout like:

```
projectA/
       conftest.py
       settings.py
       test_foo.py
projectB/
       conftest.py
       settings.py
       test_bar.py
```

With `projectA/conftest.py` containing:

```
os.environ['DJANGO_SETTINGS_MODULE'] = 'projectA.settings'
django.setup()
```

And `projectB/conftest.py` containing:

```
os.environ['DJANGO_SETTINGS_MODULE'] = 'projectB.settings'
django.setup()
```

In this setup, it is not safe to batch `test_foo.py` and `test_bar.py` into the same `pytest` process, because:

1. Each test depends on a different set of Django settings, and
2. `django.setup()` invokes `django.apps.populate()`, which is not reentrant (see code here)

The user in this scenario would need some way to mark `test_foo.py` and `test_bar.py` as unsafe to be batched together.

Process-level settings from Pants

Some existing metadata about `python_test` targets will need to be uniform for all files in a batch:

- `resolve`
- `extra_env_vars`
- `xdist_concurrency`

Additionally, the `timeout` field can vary because it will be summed.

Syntax Options

Option 1: No new syntax, use `conftest.py` scopes

`conftest.py` files are `pytest`'s standard mechanism for sharing test fixtures / config across multiple files. Pants could identify the boundaries of each `conftest.py`'s scope in the directory tree, and ensure tests in different scopes run in different batches.

This approach would have the benefit of Just Working for many cases. Unfortunately, I don't think it would stretch to cover all use-cases. `conftest.py`s are hierarchical, and files lower in the tree can override fixtures from higher in the tree. Consider a project like:

```
src/
	conftest.py
	test_foo.py
	subpackage/
		conftest.py
		test_bar.py
```

In this setup, `test_foo.py` and `test_bar.py` *might* be safe to run in the same batch, but there's no way to know without understanding the logic in each `conftest.py`. The safe thing to do would be to not batch the two, but this will result in subpar performance for many users. Ultimately I think we'd need to let users manually override which batch each conftest falls into, at which point we'd be adding a new field (maybe even a whole new target type?) - I'd then question whether the complexity is worth it over the following option.

Option 2: New field on `python_test` targets

Pants could add a new `batch_id` field to the `python_test` target. If two test modules are marked with the same batch ID, they will possibly run in the same process. Users could set IDs for entire directory trees using `__defaults__` while still having a per-file "escape hatch" for odd cases. Tests without a specified batch ID would continue to run one-file-per-process.

This would be my preferred approach - I feel it's easy to understand from the user perspective and has minimal boilerplate.

Caching

It is not possible to safely split up a batched test run into several cache results. For example, if a batch of 5 files succeeds, we cannot split that into an individual cache result per file because the tests may depend on each other in implicit ways, e.g. that the database is set up a certain way.

Instead, we can consider batching within the single batch the user told us to use. Assuming that batch ids are assigned to each file, and that the actual batches which are created are smaller than "all files with the same batch id", then a decision needs to be made about where to cut batch boundaries.

Currently the `lint`/`fmt` goals cut batch boundaries using a stable partitioning strategy which tries to avoid affecting _all_ batches when any particular batch has changed. Adopting this strategy initially will likely make sense, but it's possible that all of `lint`/`fmt`/`test` should eventually be adjusted to use a new strategy together.

Sharding

In a scenario with:

- `N` test batches
- `M` workers
- `N`*`M` test files spread evenly across all batches

The ideal outcome of `./pants test --shard=n/M` would be for each worker to run all the tests in a single batch; the worst-case would be for each worker to run 1 file from each batch. The sharding algorithm should take batch IDs into account to make the ideal case more likely, even if we'll never hit the ratios perfectly in practice.