

# Speeding up pytest runs

SF Python meetup  
April 10, 2024



James Abel



# Agenda

- pytest intro
- pytest-xdist
- CPUs
- msqLite
- pytest-fly

# pytest intro

- **The pytest framework makes it easy to write small, readable tests, and can scale to support complex functional testing for applications and libraries**
- **Simple tests:** Writing tests is straightforward with pytest because you can use Python's built-in assert statement for test conditions.
- **Detailed info on failing assert statements:** When an assert fails, pytest provides detailed context to help you understand why.
- **Auto-discovery of test modules and functions:** pytest automatically discovers tests following its conventions, so you don't need to manually register tests.
- **Fixture support:** pytest offers powerful fixture support, which is a way to provide a fixed baseline upon which tests can reliably and repeatedly execute.
- **Parameterized testing:** You can easily parameterize tests to run the same test function with different inputs.
- **Plugins:** pytest has a vast ecosystem of plugins to extend its functionality for various needs, like parallel test execution, test coverage, and more.

# Speeding up test runs

- `mock`-ing can run faster than using real services
  - AWS: `moto`
  - `awsimple` (76 tests):
    - `moto mock`: 43 sec
    - real AWS: 417 sec
    - 9.7x speedup!
  - Also, can facilitate CI
- Only re-run tests that are impacted by changed code, e.g., using `pytest-testmon`
- Only run meaningful tests (can be difficult)
- Caching (can be even more difficult ...)
- *Parallelism ...*

# pytest-xdist

- **pytest-xdist is a plugin for the pytest framework that enables you to run tests in parallel, across multiple CPUs or even across different machines.**
- **Parallel execution:** Distribute tests across multiple CPUs to speed up the execution. This is particularly beneficial for large test suites or tests that perform time-consuming operations such as accessing the network or a database.
- **Distributed testing:** Run tests in a distributed manner across multiple machines to scale the testing process horizontally. This is useful in environments where the test suite is too large for a single machine or when you want to test in different environments simultaneously.
- **Load balancing:** Dynamically allocate tests to different CPUs or machines based on their current load, ensuring an even distribution of work and optimizing the overall test execution time.
- Generally, run-time feedback may be reduced vs. regular serial run
  - Due to how pytest-xdist is implemented, the `-s/--capture=no` option does not work.
  - <https://pytest-xdist.readthedocs.io/en/latest/>
- **Requires tests be independent**
- **`-n X` or `-n auto`**
  - `-n X` explicitly defines number of workers, e.g., `-n 4` for 4 workers
  - `-n auto` tells pytest-xdist to determine the number of workers

# “Workers” vs. CPUs

- Not all “CPUs” or “Processors” are equal
  - We’re not talking about GPUs here ... (at least yet)
- Long ago a CPU was a single processor. Now, generally at least in pairs.
- “Big” Cores (Performance cores)
  - Highest Single-Threaded performance, highest power, highest super-scaler, most execution units
- “Little” Cores (Efficiency cores)
  - Good performance/Watt
- SMT (Simultaneous Multi-Threading, AKA Intel® Hyper-Threading™ or HT)
  - 2 (or more) “virtual” processors presented to the OS from one core
  - Good for workloads where OS threads/processes don’t saturate a shared compute resource such as execution units
- Some platforms will have a mix
- Generally, set the number of workers to at least the number of “performance” cores
  - Assuming no inter-test dependencies
- Most platforms are power/thermal limited

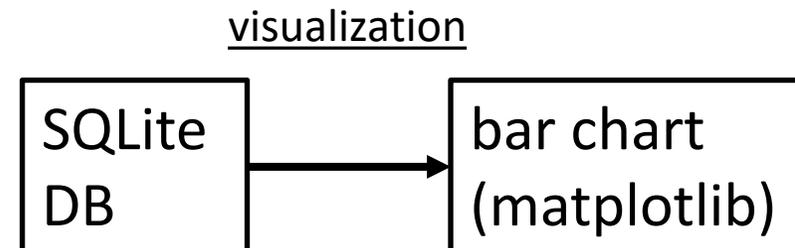
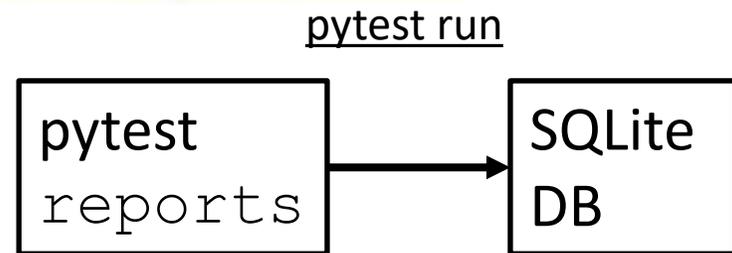
Hardware Configurations can impact Expected Performance

# pytest-fly

New!



- Enables performance rollups and visualizations of pytest runs to aid the test developer to improve test performance and reliability
- A pytest plugin that records pytest Reports to a local SQLite DB
  - Includes a basic visualization
- Installation
  - `pip install pytest-fly`
- Usage in pytest
  - `pytest --fly`
- Visualization
  - `python -m pytest_fly`



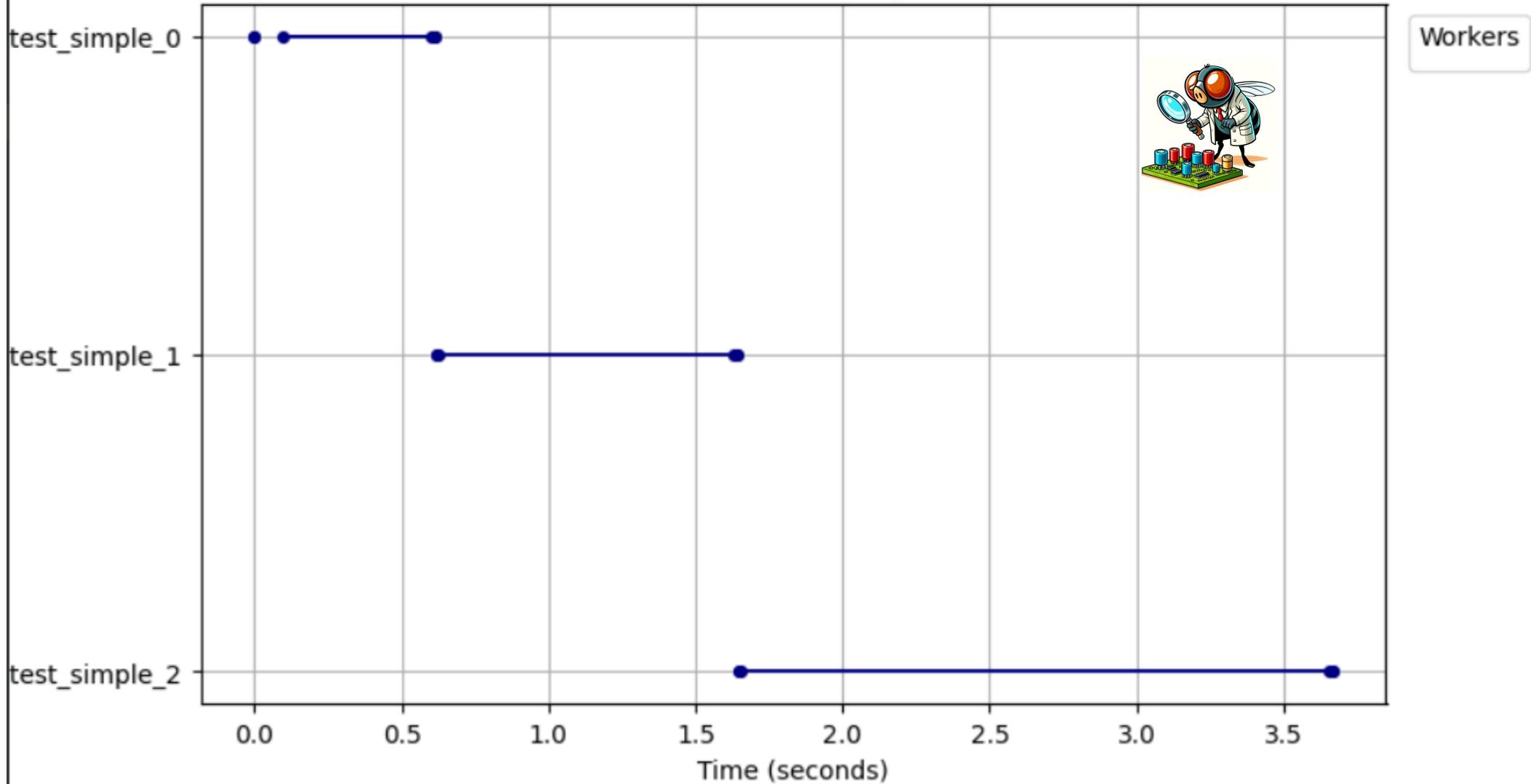
# msqlite

- multi-threaded and multi-process access to SQLite DB (file)
- A separate python package on PyPI to facilitate using SQLite DBs in a parallel environment
- Supports a subset of SQLite usage models
- Meant for infrequent, short DB writes
- Locks the DB (file) for all accesses
- Automatic retries
- `pip install msqlite`
- <https://github.com/jamesabel/msqlite>

# pytest Reports

- **nodeid**: A unique identifier for the test item. It's a string that represents the full path to the test, including the file name, class, and test function.
- **location**: A tuple containing the filesystem path to the test file, the line number where the test starts, and the test name. This provides an exact location of the test in the source code.
- **outcome**: A string indicating the outcome of the test, typically 'passed', 'failed', or 'skipped'.
- **duration**: The time taken to run the test, in seconds.
- **when**: The phase of the test execution this report represents. For TestReport, this can be 'setup', 'call', or 'teardown', indicating which phase the report is related to.

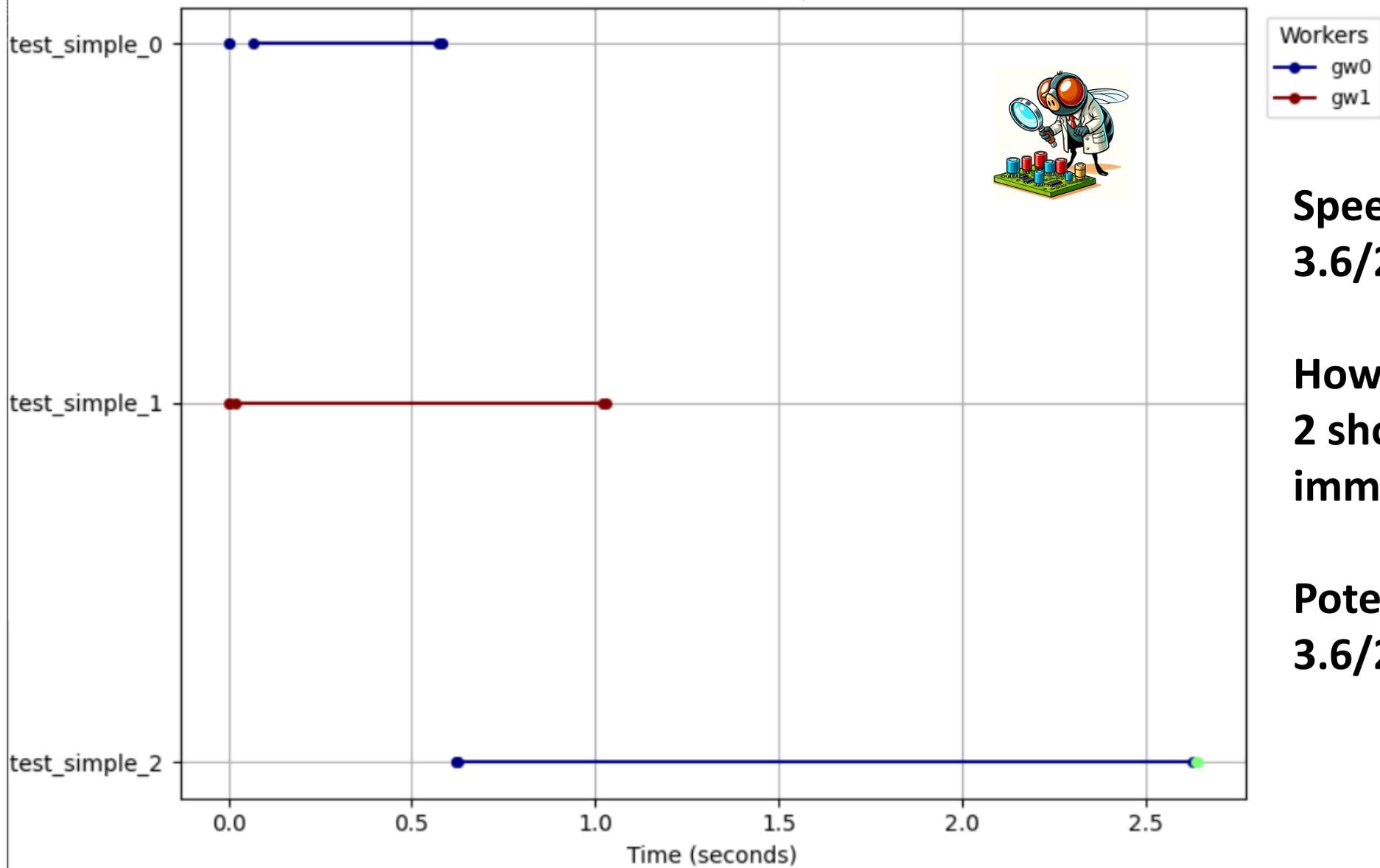
Timeline of Test Phases per Worker Overall Utilization: 95.62%



`parallel`  
`pytest-xdist`  
`-n 2`

gw1: 38.02%  
None: 0.04%  
gw0: 94.95%

Timeline of Test Phases per Worker Overall Utilization: 44.34%



**Speedup:**  
 $3.6/2.6=1.4x$

**However, test 2 should start immediately**

**Potential:**  
 $3.6/2.0=1.8x$

# awsimple (mocked)

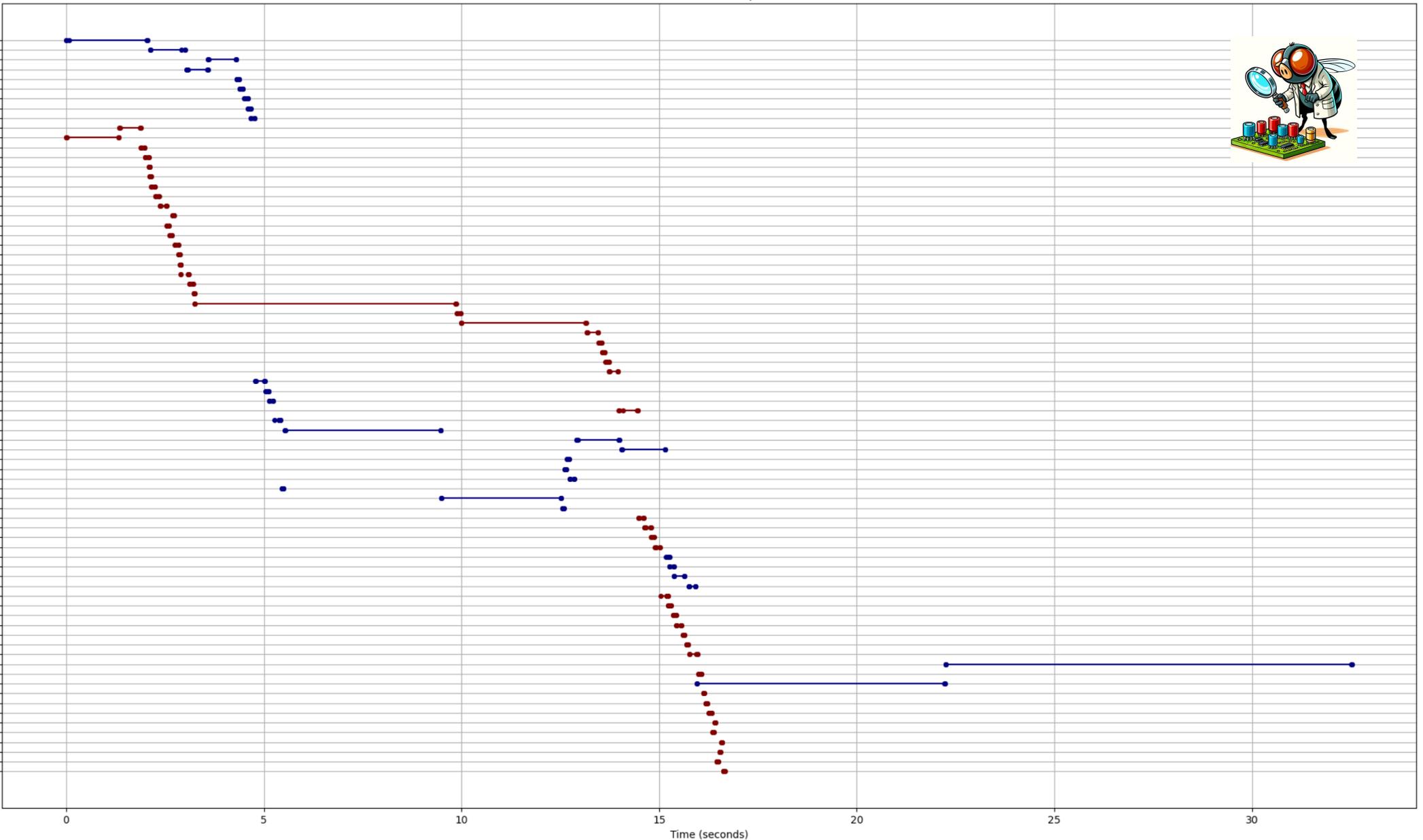
gw0: 95.46%

gw1: 44.12%

Overall Utilization: 69.79%

Timeline of Test Phases per Worker

```
awsimple/test_aws_test.py::test_aws_test
ate_table.py::test_dynamodb_create_table
simple/test_dynamodb.py::test_dynamodb
test_dynamodb.py::test_get_table_names
dynamodb_delete.py::test_dynamodb_delete
tems.py::test_dynamodb_delete_all_items
db_get_item.py::test_dynamodb_get_item
found.py::test_dynamodb_item_not_found
amodb_miv_ui.py::test_dynamodb_miv_ui
r.py::test_dynamodb_partition_as_number
mber.py::test_dynamodb_sort_as_number
ynamodb_query.py::test_dynamodb_query
kwargs.py::test_dynamodb_query_kwargs
y::test_dynamodb_scan_cache_cache_life
e.py::test_dynamodb_scan_cache_mtime
ict.py::test_dynamodb_scan_table_as_dict
index.py::test_dynamodb_secondary_index
k.py::test_dynamodb_secondary_index_int
test_dynamodb_table_not_found_get_item
test_dynamodb_table_not_found_put_item
dynamodb_table_not_found_upsert_item
ynamodb_upsert.py::test_dynamodb_upsert
st_get_account_id.py::test_get_account_id
ation_information.py::test_get_access_key
figuration_information.py::test_get_region
test_awsimple/test_logs.py::test_logs
/test_lru_cache_helpers.py::test_disk_free
cache_helpers.py::test_get_directory_size
test_awsimple/test_mock.py::test_mock
st_recent_error.py::test_most_recent_error
ysimple/test_s3_bucket.py::test_s3_bucket
t_s3_bucket.py::test_s3_bucket_not_found
t_not_found.py::test_s3_bucket_not_found
awsimple/test_s3_delete.py::test_s3_delete
test_awsimple/test_s3_dir.py::test_s3_dir
awsimple/test_s3_dir.py::test_s3_dir_prefix
ot_exist.py::test_s3_object_does_not_exist
empty_bucket.py::test_s3_empty_bucket
st_s3_file_transfer.py::test_cache_eviction
sfer.py::test_get_never_change_metadata
s3_file_transfer.py::test_s3_big_file_upload
file_transfer.py::test_s3_download_cached
transfer.py::test_s3_download_cached_dir
ile_transfer.py::test_s3_download_dest_dir
nsfer.py::test_s3_download_dest_full_path
3_metadata_not_uploaded_with_awsimple
st_s3_file_transfer.py::test_s3_read_string
le/test_s3_file_transfer.py::test_s3_upload
st_s3_file_transfer.py::test_s3_z_metadata
st_awsimple/test_s3_keys.py::test_s3_keys
mple/test_s3_keys.py::test_s3_keys_prefix
st_s3_list_buckets.py::test_s3_list_buckets
le_transfers.py::test_s3_multiple_transfers
s3_object_floats.py::test_s3_object_floats
est_s3_public_readable.py::test_s3_upload
python_object.py::test_s3_python_object
awsimple/test_s3_string.py::test_s3_string
s3_transfer_lines.py::test_s3_transfer_lines
test_serializable.py::test_make_serializable
simple/test_sns_create.py::test_sns_create
mple/test_sns_publish.py::test_sns_publish
ue.py::test_sqs_create_and_delete_queue
mple/test_sqs_get_arn.py::test_sqs_get_arn
messages.py::test_sqs_immediate_delete
t_sqs_messages.py::test_sqs_n_messages
sages.py::test_sqs_poll_immediate_delete
s_messages.py::test_sqs_poll_user_delete
y::test_sqs_message_available_and_purge
s_queue_exists.py::test_sqs_queue_exists
eive_nothing.py::test_sqs_receive_nothing
g.py::test_sqs_receive_nothing_poll_many
ing.py::test_sqs_receive_nothing_poll_one
rovided_timeout.py::test_actually_timeout
py::test_user_provided_minimum_timeout
d_timeout.py::test_user_provided_timeout
rovided_timeout_nonsensical_parameters
```



Workers  
- gw1  
- gw0

# pytest-fly next steps

- Testing of pytest-fly itself
- Improved visualization/UI
- Documentation
- Examples
- <https://github.com/jamesabel/pytest-fly>



# BACKUP

---