Key Advantages of Test Distribution Relative to Alternative Approaches

A good way to understand the advantages of Test Distribution is to understand the limitations of alternative approaches to test parallelization:

No parallelism

Not being able to parallelize your tests is a serious constraint which will become more and more costly as your test base grows, ultimately resulting in test times that take an order of magnitude longer to run compared to modern test distribution solutions.

- The size of a test set tends to grow along with its associated project, with very large or complex applications spending the majority of their time in the test phase of the build.
- Without a plan for parallelization of tests, test feedback cycle times will increase continuously as more tests are added to the set.
- New tests are necessary when new features are added, and though Test Impact Analysis can help to reduce the test set executed during each build, there are many scenarios where running tests is unavoidable.

CI Fanout

CI fanout achieves parallel execution by creating multiple CI jobs per build, where each CI job runs on a subset of the tests on a different agent. While valuable when compared to not leveraging any form of test distribution, CI fanout has many disadvantages:

- There is no aggregate test reporting since test reports are separated per agent.
- There is no efficient load balancing across agents as test partitioning must be done manually.
- Significant CI administrative overhead may result from running tens or hundreds of CI jobs for the same logical build step.
- Debugging is inefficient since the same builds are run many times, each generating its own build log.
- Feedback cycles and resource consumption may be negatively impacted by the necessity to run full builds per agent.
- CI fanout only works on CI. As a result, a pull-request has to be created just to get fast feedback on tests, negatively impacting developer productivity and the developer experience.

Single-machine parallelism

Single-machine parallelism often results in a significant acceleration, but is limited by the number of parallel forks, which is dictated by a machine's CPU and memory resources. This has several implications:

- Tests that consume a lot of local developer machine resources become unavailable for other tasks.
- Organizations invest in expensive, powerful machines to run CI agents and improve test execution time, resulting in massively underutilized machine capacity. As soon as a CI build starts, CI machine resources are fully dedicated to CI builds and are not available for other jobs.
- Resource sharing conflicts often result since single-machine parallelism may dictate running tests in multiple local JVMs to avoid interfering with each other. They share the same file system and often the same "process space" so servers started from tests need to use different ports.

Modern Test Distribution (TD)

Modern fine-grained and transparent TD solutions offer many advantages:

- TD is available for all local and CI builds regardless of where invoked (e.g., local command line, IDE).
- There is no change to the developer workflow since tests are run automatically in a distributed and transparent manner, instead of run locally.
- The CI user experience is better compared to CI fan out since there is only one build invocation and CI job. If properly implemented, test report aggregation happens as if all tests are run on one agent. There is only one build log.
- Resource consumption is dramatically reduced since the prerequisite build steps for running tests run only once.
- Reliability is increased since efficient fine-grained distribution allows for sequential execution of tests per agent.
- TD history and dynamic scheduling services can be used to balance the test execution load properly and elastically between test agents and optimize test execution time and compute resource consumption.
- To maximize available resources, TD supports local test execution fallback if no remote test agents are available or an optional hybrid local/remote execution mode.