Qiskit Experiments

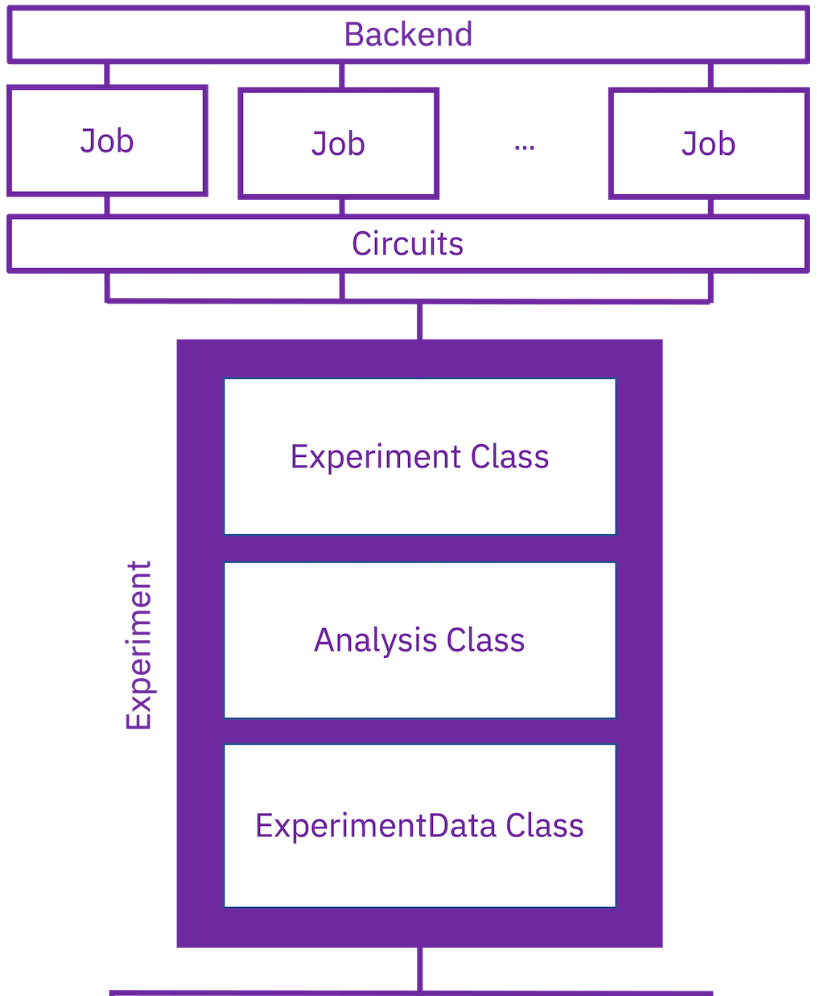
https://qiskit-extensions.github.io/qiskit-experiments

曾祥东

2024 年 3 月 1 日

简介

- 基于 Qiskit 核心功能
- 主要用于设备描述和校准实验
- 实验
  - 包括量子线路和相关元数据
  - 可在真实或模拟的量子后端上执行
  - 自动运行作业，并生成数据、拟合参数及绘图
- 实验框架：`Experiment` 和 `Analysis` 类
  - `Experiment`：解析线路、元数据和选项，打包任务后发送至指定后端设备
  - `Analysis`：处理并分析原始数据，输出分析结果、图表等



Qiskit Core (1)

- `qiskit.circuit`：量子线路
  - `QuantumCircuit`、`Qubit`、`QuantumRegister`、`Gate`、`IfElseOp` 等
- `qiskit.dagcircuit`：量子线路的有向无环图 (DAG) 表示
- `qiskit.qasm2`、`qiskit.qasm3`：`QuantumCircuit` 与 OpenQASM 的互相转换
- `qiskit.pulse`：底层实时脉冲控制
  - 脉冲：`Waveform(samples)`、`SymbolicPulse(pulse_type, duration, parameters)`
  - 信号通道（硬件通道抽象层）：`PulseChannel`、`AcquireChannel` 等
  - 指令：`Delay`、`ShiftPhase`、`Play(pulse, channel)` 等
  - 指令调度：`Schedule(Schedule|Instruction, ...)`、`ScheduleBlock`
- `qiskit.qobj`：传递给 provider 的负载 (payload)
  - `PulseQobj`、`QasmQobj` 等

Qiskit Core (2)

- `qiskit.compiler`
  - `transpile`：量子线路的转译和优化（`QuantumCircuit` → `QuantumCircuit`）
  - `assemble`：汇编器（`QuantumCircuit` → `Qobj`）
- `qiskit.primitives`：构建计算的基础单元
  - `Estimator(V2)`：预估器，接受量子线路和观测量的组合并给出期望值
  - `Sampler(V2)`：采样器，接受量子线路并对输出（经典）寄存器采样
- `qiskit.providers`
  - `Provider`：对后端的进一步封装
  - `Backend`：后端的具体实现
    - `target`：为 transpiler/compiler 定义模型
    - `run`：接受任务提交

Qiskit 与量子线路 (1)

- `QuantumCircuit` 类

```python
from qiskit import QuantumCircuit
from qiskit.circuit.library import HGate

qc = QuantumCircuit(1)
qc.x(0)
qc.append(HGate(), [0])
```

- `Operator` 类

```python
from qiskit.quantum_info.operators import Operator,

# [[0, 0, 0, 1], [0, 0, 1, 0], [0, 1, 0, 0], [1, 0,
XX = Operator(Pauli('XX'))

qc = QuantumCircuit(2, 2)
qc.append(XX, [0, 1])
qc.measure([0,1], [0,1])
```

- OpenQASM

```python
from qiskit.qasm3 import loads, dumps

# Import
program = """
    OPENQASM 3.0;
    include "stdgates.inc";

    input float[64] a;
    qubit[3] q;
    bit[2] mid;
    bit[3] out;

    ...
"""
circuit = loads(program)

# Export
qc = QuantumCircuit(2)
qc.h(0)
qc.cx(0,1)
dumps(qc)
```

Qiskit 与量子线路 (2)

- 脉冲门 (pulse gate)：量子线路的底层实现
- 校准 (calibration)：逻辑线路门 → `ScheduleBlock` 的映射

```
# 构建量子线路（Bell 态）
from qiskit import QuantumCircuit

circ = QuantumCircuit(2, 2)
circ.h(0)
circ.cx(0, 1)
circ.measure(0, 0)
circ.measure(1, 1)

# 在 q0 上为 Hadamard 门添加校准
from qiskit import pulse
from qiskit.pulse.library import Gaussian

with pulse.build(backend, name='hadamard') as h_q0:
    pulse.play(Gaussian(duration=128, amp=0.1, sigma=16), pulse.drive_channel(0))
circ.add_calibration('h', [0], h_q0)
```

Qiskit 转译器

- 转译阶段

1. 初始化 (init)：验证指令，将多比特门转换为单/双比特门
2. 布局 (layout)：将虚拟量子比特映射为物理量子比特
3. 路由 (routing)：根据硬件连接图插入 `SWAP` 门
4. 翻译 (translation)：翻译为基本指令集
    - `SWAP` → `CNOT` × 3
    - `TOFFOLI` → `CNOT` × 6
5. 优化 (optimization)：更高效地拆分和重组线路
    - O1：`Optimize1qGatesDecomposition`、`CXCancellation`
    - O2：`CommutativeCancellation`
    - O3：`Collect2qBlocks`、`ConsolidateBlocks`、`UnitarySynthesis` 等
6. 调度 (scheduling)：时序对齐、插入 `Delay` 指令等

- 统一的转译接口：`PassManager.run(circuits)`

Qiskit Experiments 架构

模块	功能
`qiskit_experiments.framework`	实验框架，提供了实验与分析的基础类
`qiskit_experiments.library`	实验库，包含各类量子表征、校准和验证实验
`qiskit_experiments.calibration_management`	管理校准实验结果数据
`qiskit_experiments.database_service`	数据库服务
`qiskit_experiments.data_processing`	实验数据处理
`qiskit_experiments.curve_analysis`	曲线拟合
`qiskit_experiments.visualization`	可视化

实验框架

- 基本流程

```
from qiskit_experiments.library import SomeExperiment

# 初始实验设定
exp = SomeExperiment(physical_qubits, **options)

# 在指定后端上运行
exp_data = exp.run(backend)

# 等待实验完成
exp_data.block_for_results()

# 保存结果至数据库（可选）
exp_data.save()

# 分析结果
for result in exp_data.analysis_results():
    print(result)
```

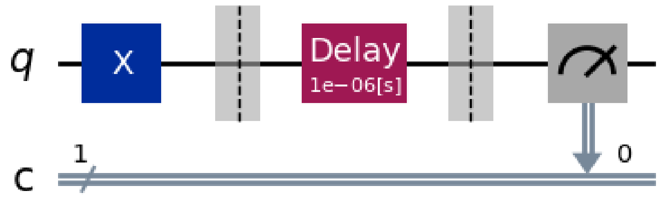
- 示例：`hello.ipynb`

实验类

- 基类：`BaseExperiment`
  - 复合实验：`BatchExperiment`、`ParallelExperiment`、`(CompositeExperiment)`
  - 验证实验：`StandardRB`、`InterleavedRB`、`TomographyExperiment` 等
  - 特征实验（单比特）：`T1`、`T2Hahn`、`T2Ramsey`、`Tphi`、`QubitSpectroscopy` 等
  - 特征实验（两比特）：`CrossResonanceHamiltonian`、`ZZRamsey`、`FineZXAmplitude` 等
  - 校准实验：`FrequencyCal`、`RoughDragCal`、`FineAmplitudeCal` 等
- 大部分实验会有对应的分析类：
  - `T1` → `T1Analysis`、`RoughDrag` → `DragCalAnalysis` 等

示例：T1 实验

- 物理背景：
  - 由于退相干 (decoherence) 机制，在一定的延迟 (delay) 之后，处于激发态的量子比特可能回到基态
  - t 时间后测得 $|1\rangle$ 的概率为 $P(|1\rangle) = Ae^{-t/T_1} + B$
- 量子线路：



- 代码：

```
import numpy as np
from qiskit_experiments.library import T1
from qiskit_ibm_runtime.fake_provider import FakePerth
from qiskit_aer import AerSimulator

# 选择后端
backend = AerSimulator.from_backend(FakePerth())

# 设置量子比特与延迟
qubit0_t1 = FakePerth().qubit_properties(0).t1
delays = np.arange(1e-6, 3 * qubit0_t1, 3e-5)

# 创建实验
exp = T1(physical_qubits=(0,), delays=delays)

# 运行实验
exp_data = exp.run(backend=backend)

# 分析结果
display(exp_data.figure(0))
```

实验类 `T1`

```python
from qiskit.circuit import QuantumCircuit
from qiskit_experiments.framework import BackendTiming, BaseExperiment, Options
from qiskit_experiments.library.characterization.analysis.t1_analysis import T1Analysis

class T1(BaseExperiment):
    def __init__(self, physical_qubits, delays, backend):
        super().__init__(physical_qubits, analysis=T1Analysis(), backend=backend)
        self.set_experiment_options(delays=delays)

    def circuits(self) -> List[QuantumCircuit]:
        timing = BackendTiming(self.backend)
        circuits = []
        for delay in self.experiment_options.delays:
            circ = QuantumCircuit(1, 1)
            circ.x(0)
            circ.barrier(0)
            circ.delay(timing.round_delay(time=delay), 0, timing.delay_unit)
            circ.barrier(0)
            circ.measure(0, 0)
            circ.metadata = {"xval": timing.delay_time(time=delay)}
            circuits.append(circ)
        return circuits
```

实验基类 `BaseExperiment` (1)

```python
class BaseExperiment(ABC, StoreInitArgs):
    def run(self, backend, analysis, timeout, **run_options) -> ExperimentData:

        # Make a copy to update analysis or backend if one is provided at runtime
        experiment = self.copy()
        experiment._set_backend(backend)
        experiment._finalize()

        # Generate and transpile circuits
        transpiled_circuits = experiment._transpiled_circuits()

        # Initialize result container
        experiment_data = experiment._initialize_experiment_data()

        # Run options
        run_opts = experiment.run_options.__dict__

        # Run jobs
        jobs = experiment._run_jobs(transpiled_circuits, **run_opts)
        experiment_data.add_jobs(jobs, timeout=timeout)

        return experiment.analysis.run(experiment_data)
```

实验基类 `BaseExperiment` (2)

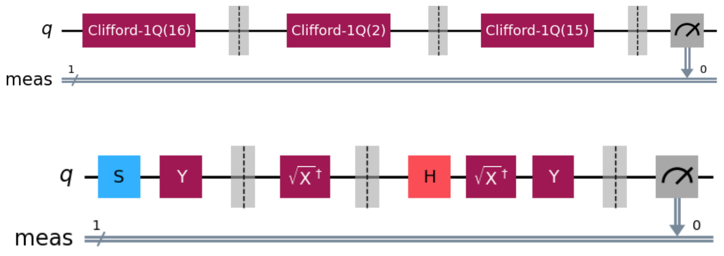
```python
from qiskit import transpile, QuantumCircuit

class BaseExperiment(ABC, StoreInitArgs):
    def _transpiled_circuits(self) -> List[QuantumCircuit]:
        transpile_opts = copy.copy(self.transpile_options.__dict__)
        transpile_opts["initial_layout"] = list(self.physical_qubits)
        transpiled = transpile(self.circuits(), self.backend, **transpile_opts)

        return transpiled

    def _run_jobs(self, circuits: List[QuantumCircuit], **run_options) -> List[Job]:
        max_circuits = self._max_circuits(self.backend)
        job_circuits = [circuits[i : i + max_circuits] for i in range(0, len(circuits), max_circuits)]

        # Run jobs
        jobs = [self.backend.run(circs, **run_options) for circs in job_circuits]

        return jobs
```

分析类 `T1Analysis`

```python
import qiskit_experiments.curve_analysis as curve

class T1Analysis(curve.DecayAnalysis):

    def _evaluate_quality(self, fit_data: curve.CurveFitResult) -> Union[str, None]:
        """Algorithmic criteria for whether the fit is good or bad."""
        amp = fit_data.ufloat_params["amp"]
        tau = fit_data.ufloat_params["tau"]
        base = fit_data.ufloat_params["base"]

        criteria = [
            0 < fit_data.reduced_chisq < 3,
            abs(amp.nominal_value - 1.0) < 0.1,
            abs(base.nominal_value) < 0.1,
            curve.utils.is_error_not_significant(amp, absolute=0.1),
            curve.utils.is_error_not_significant(tau),
            curve.utils.is_error_not_significant(base, absolute=0.1),
        ]

        if all(criteria):
            return "good"
        return "bad"
```

示例：RB 实验

- 物理背景：
  - 生成随机的 Clifford 线路
  - 通过计算 ERC (Error Per Clifford) 可以估计量子设备的误差
  - 可插入额外的门 (interleaved gates) 以估计特定门的误差
- 量子线路：



- 代码：

```
lengths = np.arange(1, 800, 200)
num_samples = 10
seed = 1010
qubits = [0]

# Run an RB experiment on qubit 0
exp = StandardRB(qubits, lengths,
  num_samples=num_samples,
  seed=seed)
expdata = exp.run(backend).block_for_results()
results = expdata.analysis_results()

# The interleaved gate is the CX gate
int_exp = InterleavedRB(circuits.CXGate(),
  qubits, lengths,
  num_samples=num_samples,
  seed=seed)
int_expdata = int_exp.run(backend).block_for_results()
int_results = int_expdata.analysis_results()
```

实验类 `StandardRB`

```python
class StandardRB(BaseExperiment, RestlessMixin):
    def circuits(self) -> List[QuantumCircuit]:
        sequences = self._sample_sequences()
        circuits = self._sequences_to_circuits(sequences)
        for circ, seq in zip(circuits, sequences):
            circ.metadata = { "xval": len(seq), "group": "Clifford" }
        return circuits

    def _sample_sequences(self) -> List[Sequence[SequenceElementType]]:
        rng = default_rng(seed=self.experiment_options.seed)
        sequences = []
        for _ in range(self.experiment_options.num_samples):
            for length in self.experiment_options.lengths:
                sequences.append(self.__sample_sequence(length, rng))
        return sequences

    def __sample_sequence(self, length: int, rng: Generator) -> Sequence[SequenceElementType]:
        if self.num_qubits == 1:
            return rng.integers(CliffordUtils.NUM_CLIFFORD_1_QUBIT, size=length)
        if self.num_qubits == 2:
            return rng.integers(CliffordUtils.NUM_CLIFFORD_2_QUBIT, size=length)
        return [random_clifford(self.num_qubits, rng) for _ in range(length)]
```