Quick Guide

Institute for Quantum Computing, Baidu.

January 27, 2021

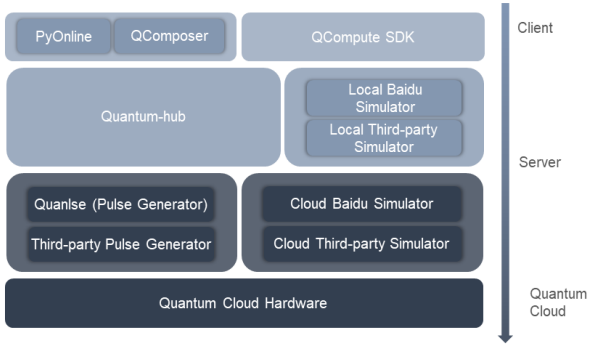
1 Overview

Introduction

Quantum Leaf is a cloud-native quantum computing platform developed by the Institute for Quantum Computing, Baidu, which provides a quantum computing environment based on the Quantum infrastructure as a Service (short for QaaS). Users can use QCompute locally or remotely by establishing a connection with an online simulator, while PyOnline and QComposer need to log in to Quantum-hub to run online.

Architecture of Quantum Leaf

Framework



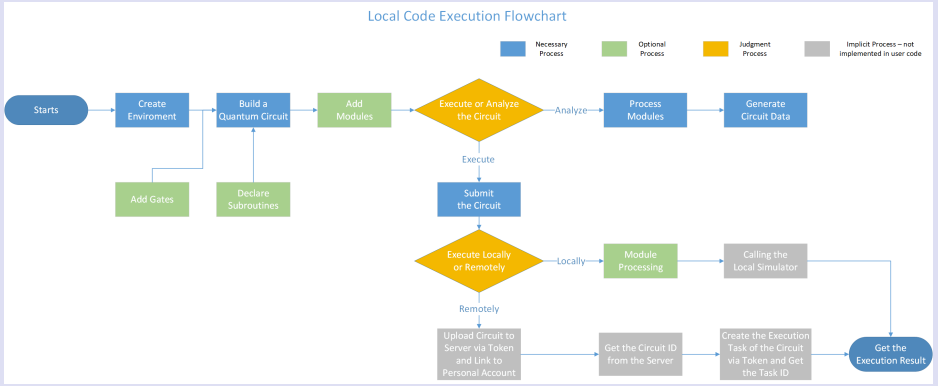
1. Three types of components are available as the front-end toolkits,
   (a) The flexible QCompute
   Users need to download and install QCompute and configure the corresponding environment. The key difference is that QCompute carries a local simulator to support local execution.
   (b) The convenient PyOnline
   Users can use hybrid programming, Python and QASM language, on PyOnline and it has many quantum circuit templates. Meanwhile, its highly efficient extendable architecture enable users upload self-defined files, download generated file, and define the installation packages.
   (c) The intuitive QComposer
   QComposer allows users constructing quantum circuits visually by “drag&drop" or performing QASM interactive programming.
2. There are local simulator, online simulator and QPU in quantum-end,
   (a) Users can use the local simulator with QCompute.
   (b) Online simulators are also available with the Token verification process. Users need to pass the Token parameters via QCompute which can be received from Quantum-hub website, in spite of the token working for PyOnline and QComposer users has been automatically invoked by Quantum-hub.
   (c) QPU is the common interface to call third-party quantum computers. VIP has more benefits, such as using the QPU for testing purpose.
   (d) The available quantum-ends of this platform are shown in the table below:

Architecture of Quantum Leaf

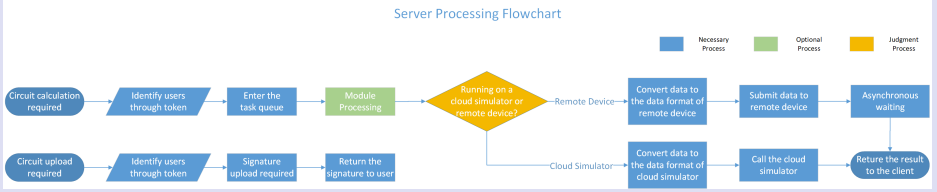
Type	Quantum End	Description
Local	LocalBaiduSim2	Sim2, local simulator, Python version
Cloud	CloudBaiduSim2	Sim2, cloud simulator (obsolete backend name)
Cloud	CloudBaiduSim2Water	Sim2, cloud simulator, Python version, dockered multiple-instances
Cloud	CloudBaiduSim2Earth	Sim2, cloud simulator, Python version, single-instance, high-performance
Cloud	CloudBaiduSim2Thunder	Sim2, cloud simulator, C++ version, single-instance, high-performance
Cloud	CloudBaiduSim2Heaven	Sim2, cloud simulator, MPI version, single-instance, High Performance Computing Cluster
Cloud	CloudAerAtBD	Aer, cloud simulator, C++ version, single-instance, high-performance
Cloud	CloudQpu	Third-party Quantum Processor Unit, only for VIPs

Flowcharts

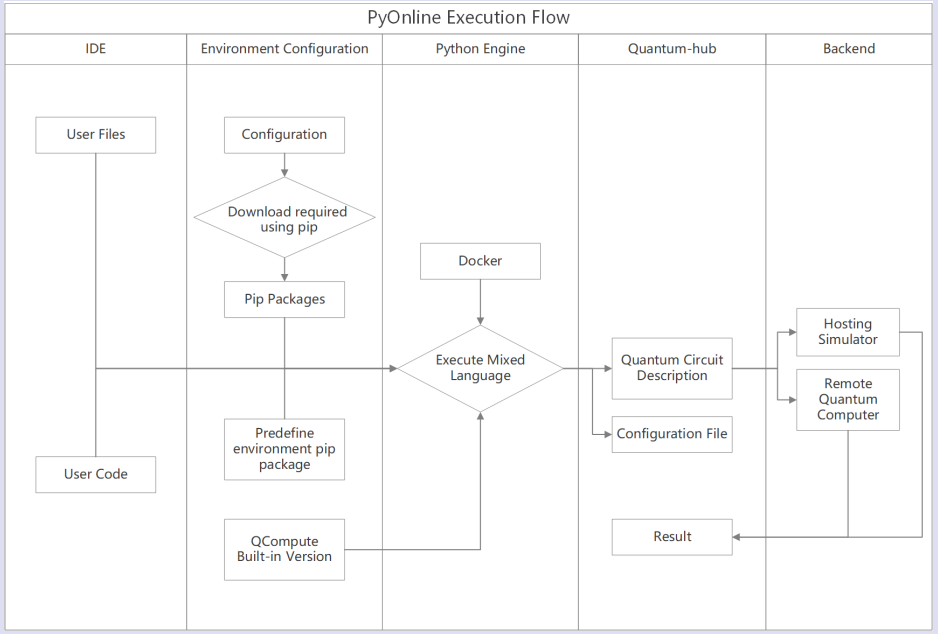
QCompute Local Code Execution Flowchart



QCompute Server Processing Flowchart



PyOnline Execution Flow



2 Example

Editors' Word

Refer to QCompute for using QCompute. The corresponding sample code is also available with the installation package which can be downloaded from github.
If confused about the basic operations and functions of PyOnline and QComposer, users can check the [Help] on the corresponding page, or ask for help through [Feedback] or QQ Group (1147781135).

PyOnline

```
from QCompute import *

# Create a new quantum environment
env = QuantumEnvironment()
# Choose a quantum end,
# the quantum simulator developed by Baidu is used in this case
env.backend(BackendName.CloudBaiduSim2)

# Initialize six qubits at state|000000>
q = [env.Q[i] for i in range(6)]
# The subscript of the qubit starts from 0,
# and add the Hadamard gate to qubit 2
H(q[2])
# Add CNOT gate with qubit 2 and qubit 4
# being the control qubit and the target qubit, respectively
CX(q[2], q[4])

# Measure qubits individually with the function MeasureZ()
# Measure q[2], q[4], q[5] respectively,
# range(3) means the number of qubits being measured is 3
MeasureZ([q[2], q[4], q[5]], range(3))
# Set the shot to 2048 means measure the circuit for 2048 times
# Set the downloadResult to False means the result does not return,
# please view it in [View Circuits] and [View Task]
taskResult = env.commit(2048, downloadResult=False)
```

Output: "011":1038 "000":1010
Note: The little-endian marking method is used for output. Therefore, the output order of the results of this example is q[5] q[4] q[2].

QComposer

Note: Using // for line comments in QComposer.

```
OPENQASM 2.0;
include "qelib1.inc";
// Initialize 6 qubits and the corresponding 6 classical bits
qreg q[6];
creg c[6];
// Add the Hadamard gate to qubit 0
h q[0];
// Add CNOT gate with qubit 0 and qubit 1
// being the control qubit and the target qubit, respectively
cx q[0], q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];
```

Output: "11":515 "00":509
QComposer realizes the visualization of quantum programming. The above code segment can be constructed with "drag&drop" which can help the user to avoid coding QASM language. The functions of the subroutines function shown below can establish a landscape of the quantum circuit with little coding.
Subroutine plays one of the key role in QComposer, making the use of parameters and quantum registers in quantum programming in a more flexible way. The use of subroutines will increase convenience for coding quantum program. For example, the following code segment shows a "Hello World".

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[6];
creg c[6];
// Declare the subroutine nG0,
// which has 3 parameters param0, param1, param2,
// specify its operation on qb0, qb1, and qb2
gate nG0(param0, param1, param2) qb0,qb1,qb2
{
    // The rzz gate uses parameters param0, param1, param2
    // and acts on the qubits specified by qb0 and qb1
    rzz(param0, param1, param2) qb0, qb1;
}
gate nG1(param0, param1, param2) qb0,qb1,qb2
{
    // Call the subroutine nG0,
    // and use the parameters with four arithmetic operations.
    nG0(param0/2, param1/4, sin(param2))  qb0, qb1, qb2;
}
// Call subroutine nG1, set param0 = param1 = param2 = pi,
// specify its operation to act on q[0], q[1], q[2]
nG1(pi, pi, pi)  q[0], q[1], q[2];
measure q[0] -> c[0];
measure q[1] -> c[1];
```

Output: "10":506 "00":518

QComposer

Here, we clarify the rules and cautions.

1. Usage Rules

   (a) There are two parts in Subroutine: declaration and call. Declaration must precede call.

   (b) When a subroutine is declared, the code block begins with the keyword gate, followed by the subroutine name, parentheses () and the quantum register. The quantum circuit is built by dragging the quantum gate icon or writing QASM code.

   (c) Any incoming parameters should be declared in parentheses, and these parameter names are placeholders for the incoming parameters. When the parameter name in parentheses is empty, QComposer will automatically complete as gate name(param0) q, but this does not affect the use of subroutine, that is, the parameter name of the declaration subroutine and the incoming parameter of the call subroutine can be empty at the same time; When the parameter name is not null, the number of parameters passed into the subroutine when calling should be the same as the number of subroutine parameters.

   (d) qb0,qb1,qb2 are placeholders for quantum registers. The quantum register to be operated on must be specified when the subroutine is called. For example, nG1(pi, pi, pi) q[0], q[1], q[2] dictates the nG1 is applied to q[0],q[1],q[2].

2. Cautions

```
gate nG1(param0, param1, param2) qb0,qb1,qb2
{
  nG0(param0/2, param1/4, sin(param2))  qb0, qb1, qb2;
}
```

   In the subrouti ne declaration code of nG1, the subroutine nG1 calls nG0. It is illegal to change the quantum gate of nG0 in this code block. Users can perform the four arithmetic operations on the nG0 parameters or set the default values, but cannot perform the four arithmetic operations on the nG1 parameters or set the default values. From this summary:

   (a) Measurement cannot be performed in the subroutine.

   (b) The quantum gate of the subroutine can be modified only in the declaration part, and the quantum gate in the called subroutine cannot be modified during the calling phase.

   (c) The four arithmetic operations or default values can be set for the parameter part of the called subroutine only in the subroutine calling part. The four arithmetic operations or default values cannot be set for the subroutine parameters in the declaration phase.

   (d) The flexibility of using parameters and quantum registers with subroutines can be shown with the following segment.

```
gate nG1(param0, param1, param2) qb0,qb1,qb2
{
    // Variable locations of parameters and quantum registers
    nG0(param2/2, param0/4, sin(param1))  qb1, qb0, qb2;
}
```

FAQ

1. How to program on Quantum Leaf?

   (a) Users can use hybrid language, Python and QASM, for experiments and development.

   (b) PyOnline and QComposer will invoke Token automatically and it is unnecessary for the users to fill in the Token by themselves.

   (c) QCompute requires transmitting the Token to connect to the remote simulator or device.

2. Why QCompute?

   (a) Hybrid language programming.

   (b) Local simualtor can work without network.

3. What are the differences between local simulator and on-line simulator?

   (a) Local simulator uses local resources, while online simulator uses computing resources from Baidu Cloud.

   (b) The computing ability of online simulator is more powerful.

4. How to accelerate the computing process?

   (a) Use online simulators.

   (b) Concurrent computing method is recommended for interactive computing.

5. Credits.

   (a) Online resource computing will consume 1 Credit every submitting and PyOnline consumes 1 extra Credit.

   (b) Please inform quantum@baidu.com or [feedback] to get more Credits.

6. How to subscribe as a VIP?

   Users can subscribe as a VIP with an invitation after going through the following steps:

   (a) Verification of corresponding identity.

   (b) Submit VIP subscribing application to: quantum@baidu.com or [feedback] in Quantum-hub.

   (c) The application will be approved after confirmation.

7. What's the difference and their connections between Quantum-hub and QCompute?

   (a) Quantum-hub is a central console of Quantum Leaf. The results can be obtained via Quantum-hub submitting from PyOnline, QComposer or QCompute.

   (b) QCompute, a high-level programming front-end of Quantum Leaf, allows users to do hybrid language development.