start

October 24, 2019

In [4]:
```
import numpy as np
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import execute

#Building circuits

#create a quantum register with 3 qubits
q=QuantumRegister(3,'q')
#create quantum circuit acting on q register
circ=QuantumCircuit(q)
# Had on q0
circ.h(q[0])
# add CX (cnot) gate on Control q0 => target q1, putting the qubits in Bell state
circ.cx(q[0],q[1])
# add CX (cnot) gate on control q0 => target q2, putting the qubits in GHZ state
circ.cx(q[0],q[2])

#vizulalize the circuit
circ.draw()
```

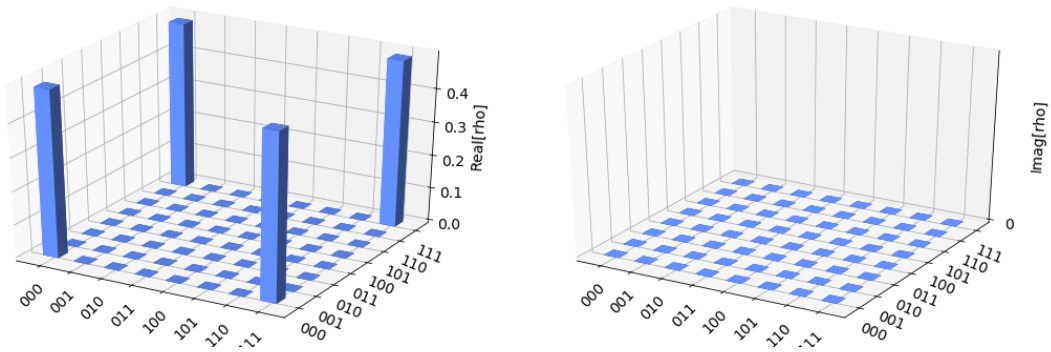
Out[4]:
```
<qiskit.tools.visualization._text.TextDrawing at 0x7f614ff72f60>
```

In [5]:
```
#Simulating circuits with Aer - statevector simulator
from qiskit import BasicAer
#run the circ on backend=statevect sim
backend=BasicAer.get_backend('statevector_simulator')
#compile and execute circuit
job=execute(circ,backend)
#take the result method
result=job.result()
#get the output -vs- statevector of the quantum circuit
outputstate=result.get_statevector(circ, decimals=3)
print(outputstate)
```

```
[0.707+0.j 0.   +0.j 0.   +0.j 0.   +0.j 0.   +0.j 0.   +0.j 0.   +0.j
 0.707+0.j]
```

In [6]:
```
#visualization toolbox
from qiskit.tools.visualization import plot_state_city
plot_state_city(outputstate)
```

Out[6]:



In [7]:
```
#simulating with open QASM backend; for this you have to add measuring AND qasm_simula
# Create a Classical Register with 3 bits.
c = ClassicalRegister(3, 'c')
# Create a Quantum Circuit
meas = QuantumCircuit(q, c)
meas.barrier(q)
# map the quantum measurement to the classical bits
meas.measure(q,c)

# The Qiskit circuit object supports composition using
# the addition operator.
qc = circ+meas

#drawing the circuit
qc.draw()
```

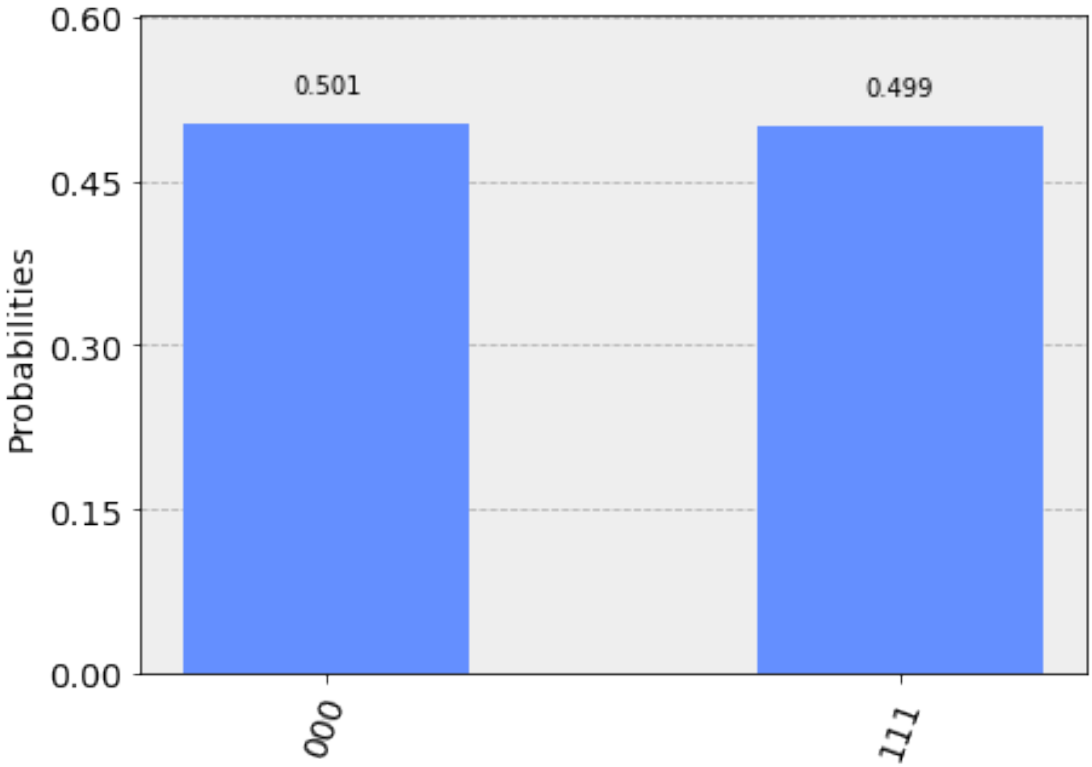
Out[7]: <qiskit.tools.visualization._text.TextDrawing at 0x7f614de150f0>

In [10]:
```
# Use Aer's qasm_simulator
backend_sim = BasicAer.get_backend('qasm_simulator')

# Execute the circuit on the qasm simulator.
# We've set the number of repeats of the circuit
# to be 1024, which is the default.
job_sim = execute(qc, backend_sim, shots=1024)

# Grab the results from the job.
result_sim = job_sim.result()
#get the aggregated binary outcomes of the circuit via get_counts
```

```
counts = result_sim.get_counts(qc)
print(counts)
#and show a hystogram
from qiskit.tools.visualization import plot_histogram
plot_histogram(counts)
```

```
{'000': 513, '111': 511}
```

Out[10]:



In [11]:
```
#now it's time for real HW
from qiskit import IBMQ
IBMQ.load_accounts()
print("Available backends:")
IBMQ.backends()
```

```
Available backends:
```

Out[11]:
```
[<IBMQBackend('ibmqx4') from IBMQ()>,
 <IBMQBackend('ibmqx2') from IBMQ()>,
 <IBMQBackend('ibmq_16_melbourne') from IBMQ()>,
 <IBMQBackend('ibmq_qasm_simulator') from IBMQ()>]
```

In [12]:
```
#choose a device having least busy queue and which it can support our 3 qubits
from qiskit.providers.ibmq import least_busy
large_enough_devices = IBMQ.backends(filters=lambda x: x.configuration().n_qubits > 4
                                                       not x.configuration().simulato
backend=least_busy(large_enough_devices)
print('best backend=',backend.name())
```

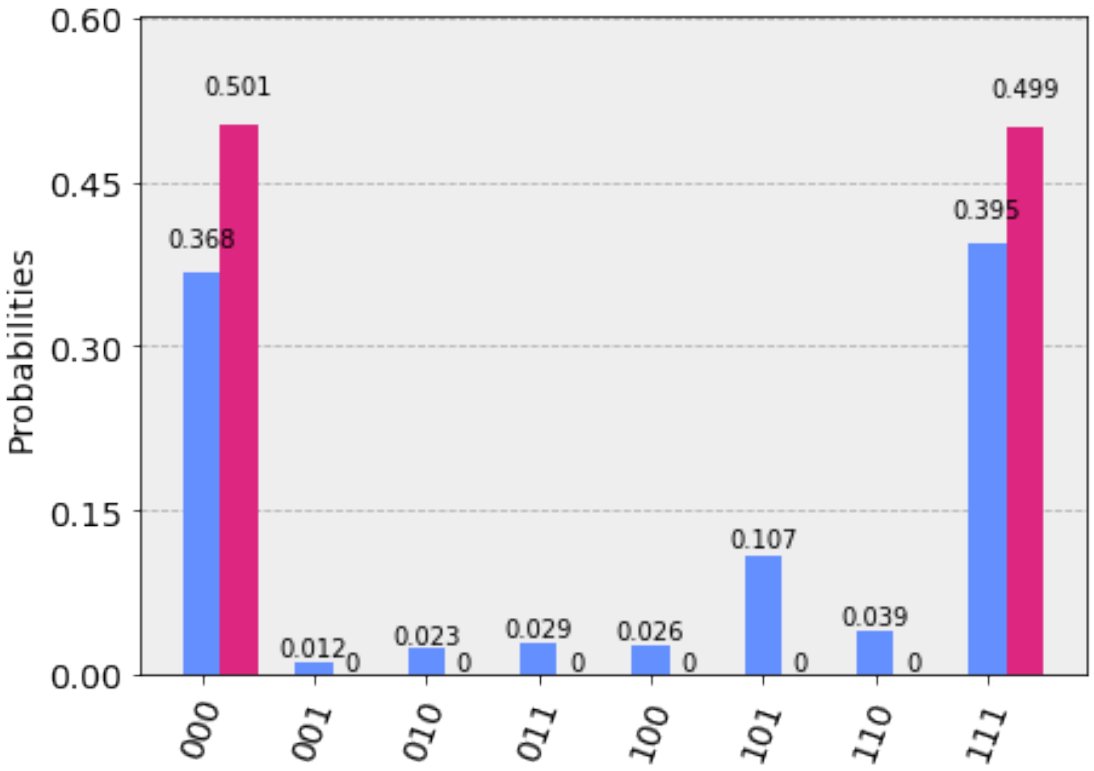
```
best backend= ibmqx4
```

In [13]:
```
#in order to run on real backend we need to specify: number of shots AND number of cr
from qiskit.tools.monitor import job_monitor
shots=1024  #number of shots running the experiment- MAX is = 8192
max_credits=3
job_exp=execute(qc,backend=backend,shots=shots,max_credits=max_credits)
job_monitor(job_exp)
```

```
HTML(value="<p style='font-size:16px;'>Job Status: job is being initialized </p>")
```

In [14]:
```
result_exp=job_exp.result()
counts_exp=result_exp.get_counts(qc)
plot_histogram([counts_exp,counts])
```

Out[14]:



In [16]:
```
#retrive the jobs - if it takes too long
jobID=job_exp.job_id()
print('job ID=',jobID)
#and having the ID the job object can be later reconstructed from backend using retri
job_get=backend.retrieve_job(jobID)
#and get results from recosntructed
job_get.result().get_counts(qc)
```

```
job ID= 5c97790d1c18ab006fb9ca42
```

Out[16]:
```
{'010': 24,
 '000': 377,
 '101': 110,
 '011': 30,
 '110': 40,
 '001': 12,
 '111': 404,
 '100': 27}
```

In [ ]: