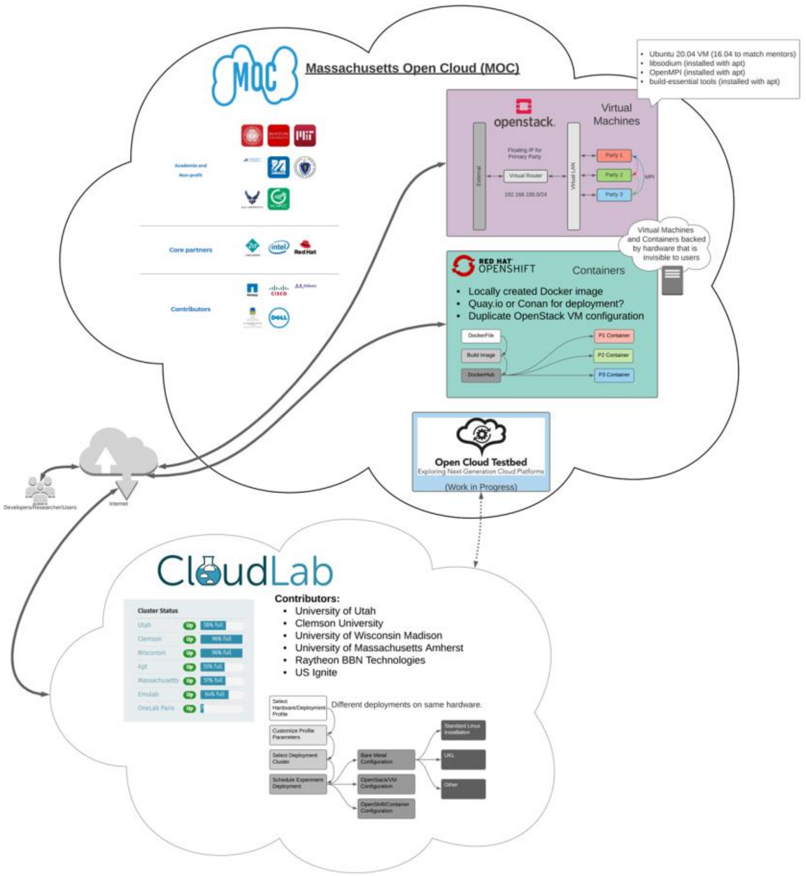
Secure Multiparty Computation Final Report

Developer | Hasnain Abdur Rehman| hasnain@bu.edu
Developer | Pierre-François Wolfe | pwolfe@bu.edu
Developer | Samyak Jain | samyakj@bu.edu
Developer | Suli Hu | sulihu@bu.edu
Developer | Yufeng Lin | yflin@bu.edu
Mentor/Client | John Liagouris | liagos@bu.edu
Mentor/Client | Vasiliki Kalavri | vkalavri@bu.edu
Subject-Matter Expert | Mayank Varia | varia@bu.edu

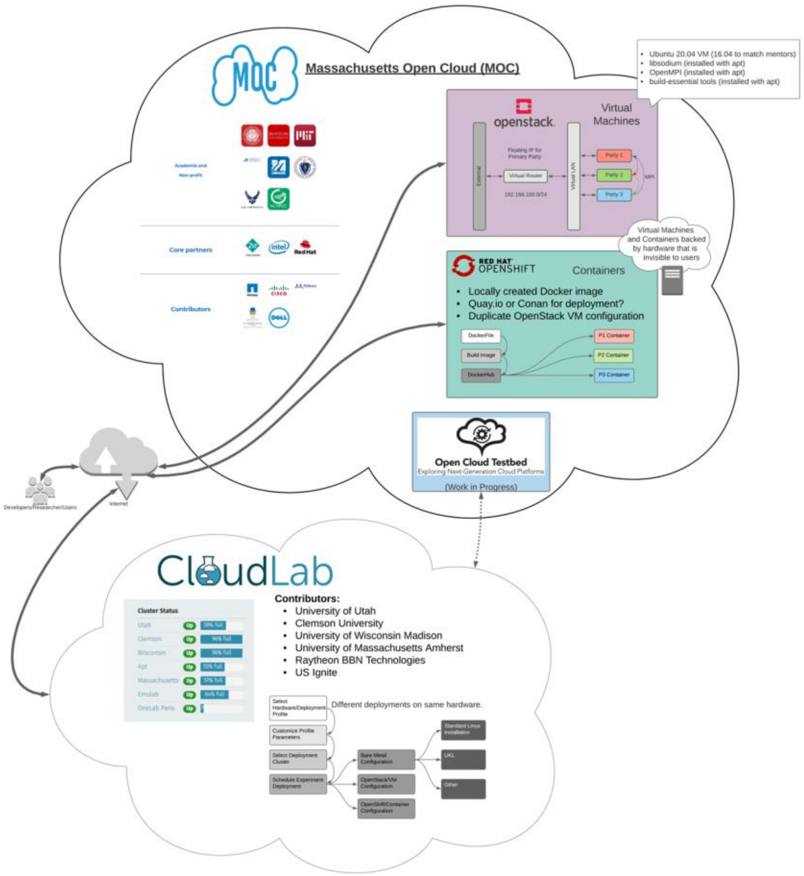
Presentation Outline

- Overview
- Deployment Environments
- Testing Instrumentation
- Data Analysis
- Automation/Replicability
- Conclusions & Future Work



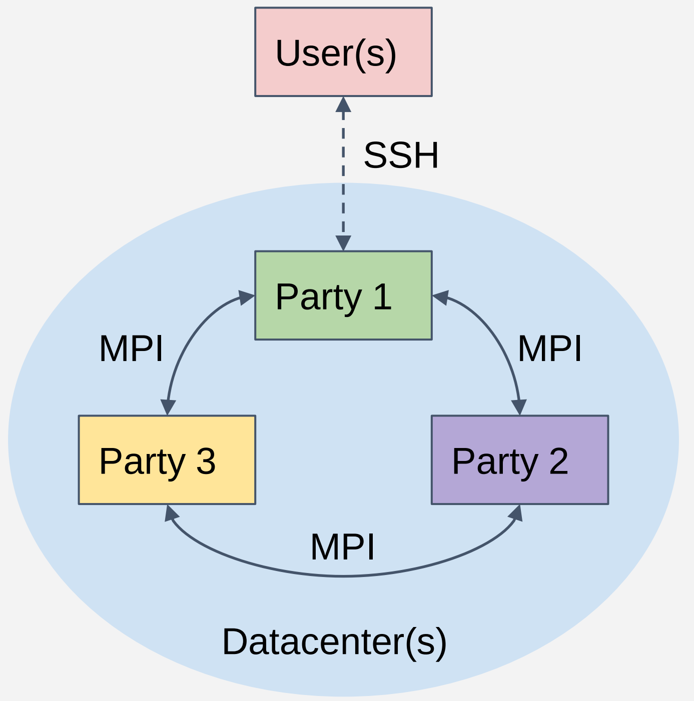
Presentation Outline

- Overview
- Deployment Environments
- Testing Instrumentation
- Data Analysis
- Automation/Replicability
- Conclusions & Future Work



Motivation

- MPC enables...
  - Shared Computation on Private Data
  - Protects the Privacy of Data
  - Mutually Agreed Computation
- Our mentors…
  - Use three party Secret Sharing MPC
  - Perform Database Queries with MPC
  - Keep all operations secure (under MPC) insecure steps

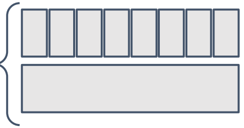


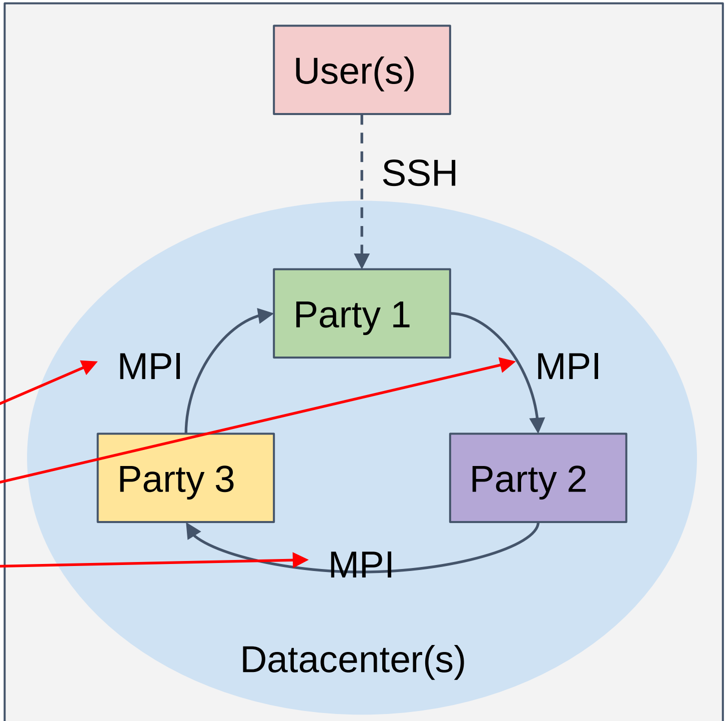
Example MPC Arrangement

Term: Multi-Party Computation (MPC)

MPC Data Exchange Primitives

- Secret Sharing MPC
  - Latency Bounded
  - Ring communication pattern
  - Each party passes data clockwise
- MPI Communication
  - Sync/Async?
  - Serial/Batch?
  - Quantity of messages?
  - Other?...





Example MPC Arrangement

Project Goals

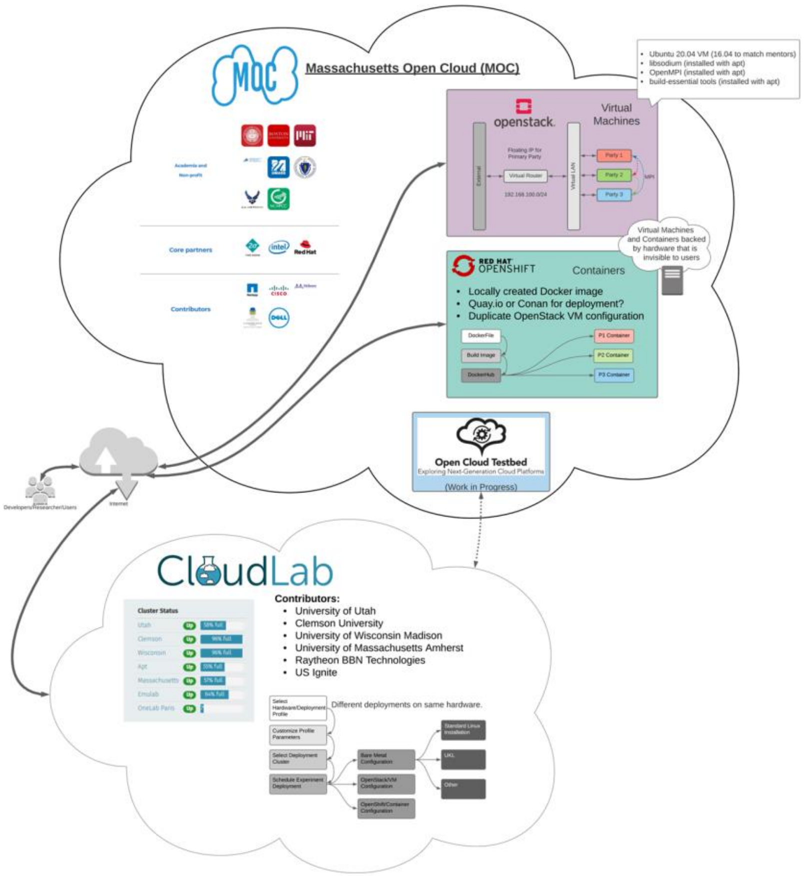
- Deployment
  - Run MPC library in multiple environments
- Testing
  - Create test scenarios conducive to analysis
  - Improve instrumentation of MPC experiments
  - Benchmark the networking layer
  - Analyze and gain initial insights
- Automation
  - Make ongoing testing easy and repeatable

Accomplishments

- Deploy to:
  - CloudLab bare-metal (various hardware, clusters, topologies)
  - MOC OpenStack (VMs)
  - MOC OpenShift (containers)
- Testing (Profiling)
  - Improve exp-exchange testbench (in source code)
  - Collect benchmark and trace data with Score-P
  - Inspect with CUBE GUI (and others)
- Automation
  - Manual config → Shell scripts → Ansible

Presentation Outline

- Overview
- **Deployment Environments**
- Testing Instrumentation
- Data Analysis
- Automation/Replicability
- Conclusions & Future Work



OpenStack

- Single VM to determine setup manually
- Some shell scripts to help automate setup
- One reference VM (semi-automated)
  - Snapshot and duplicate for other parties
- Troubleshoot SSH communication between VMs

CloudLab

- Initial test with predefined scripts
  - Testing with the provided openstack image
- Custom scripts
  - What went wrong, then what did we learn?
- Compare some different topologies
  - LAN vs RING
  - Single vs multi cluster
  - Keep a local or VM result for reference?

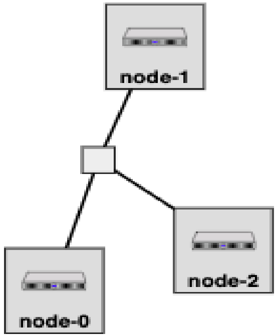
BOSTON UNIVERSITY

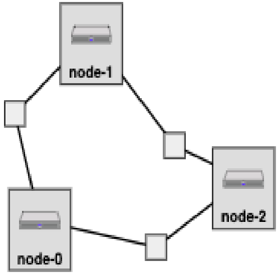
Cloudlab Topologies

LAN

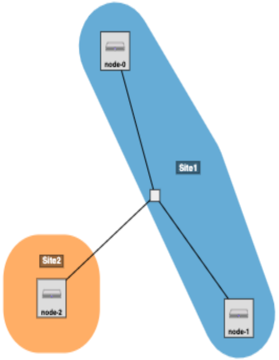
Ring

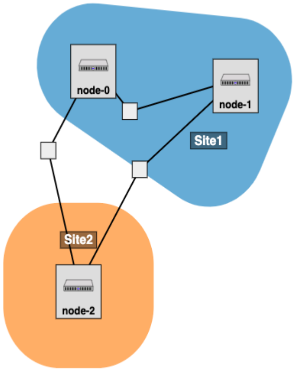
Single Cluster





Multi Cluster





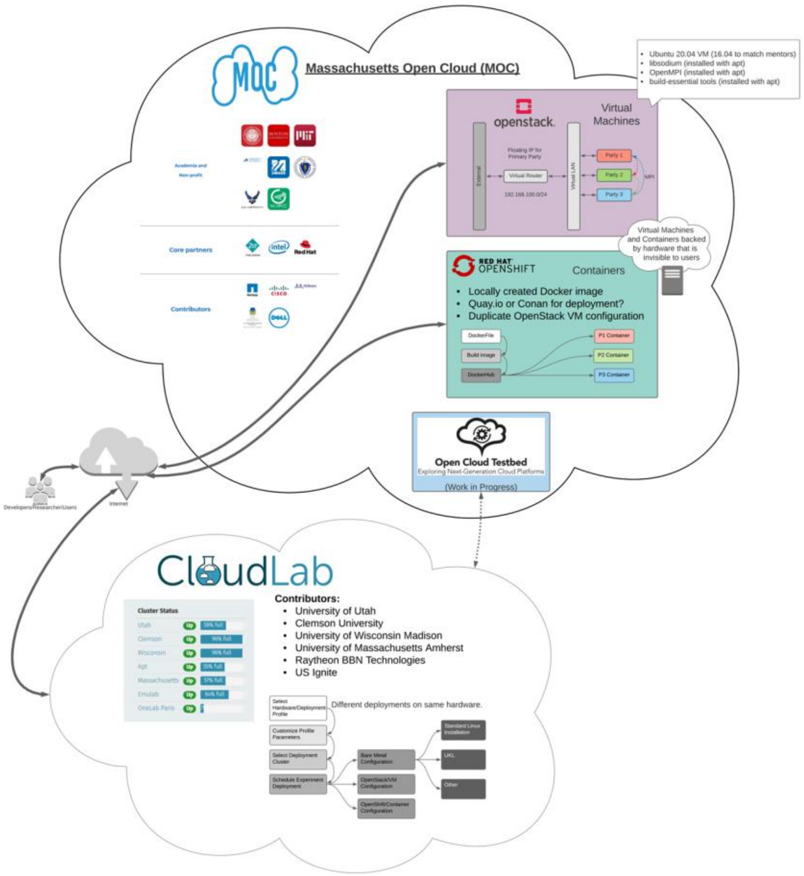

```python
"""ubuntu baremetal multi-site LAN"""

# Import the Portal object.
import geni.portal as portal
# Import the ProtoGENI library.
import geni.rspec.pg as pg
# Import the Emulab specific extensions.
import geni.rspec.emulab as emulab

pc = portal.Context()

pc.defineParameter("node_type_1", "Hardware Type for Site 1",
                   portal.ParameterType.NODETYPE, "any")
pc.defineParameter("node_type_2", "Hardware Type for Site 2",
                   portal.ParameterType.NODETYPE, "any")
pc.defineParameter("node_count", "Number of Machines",
                   portal.ParameterType.INTEGER, 3)

params = pc.bindParameters()
request = portal.context.makeRequestRSpec()

node = []

# Create selected number of nodes
for i in range(params.node_count):
    node.append(request.RawPC('node-%d' % i))
    node[i].disk_image = 'urn:publicid:IDN+emulab.net+image+emulab-ops:UBUNTU16-64-STD'
    if (i < params.node_count - 1):     #Condition can be changed based on requirement
        node[i].Site("Site1")
        node[i].hardware_type = params.node_type_1
    else:
        node[i].Site("Site2")
        node[i].hardware_type = params.node_type_2

# Create a LAN for all the connections
lan = request.LAN("lan")
lan.bandwidth = 100000

# Create a link between each of the nodes to make a ring
for i in range(params.node_count):
    iface = node[i].addInterface("eth1")
    iface.addAddress(pg.IPv4Address("192.168.1."+str(i+1), "255.255.255.0"))
    lan.addInterface(iface)

# Print the generated rspec
pc.printRequestRSpec(request)
```

Containers

- Initial attempt: Using a C/C++ package manager to build the app, and containerize using OpenShift s2i tool .
- Docker deployment using Dockerfiles + Docker Compose.
  - Set up SSH
  - Install and test MPI + MPC codebase.
- OpenShift
  - Unexpected behavior, unlike Docker containers
  - SSH issue
  - Permissions issues: MPI failing to launch required dependencies, create new files.

Presentation Outline

- Overview
- Deployment Environments
- **Testing Instrumentation**
- Data Analysis
- Automation/Replicability
- Conclusions & Future Work



Instrumenting the Exchange Primitive Test

Before

```c
int main(int argc, char** argv) {

  if (argc < 2) {
    printf("\n\nUsage: %s [INPUT_SIZE]\n\n", argv[0]);
    return -1;
  }
```

```c
/* ======================================================
  1. Measure exchange_array
====================================================== */
// start timer
gettimeofday(&begin, 0);

exchange_shares_array(r1s1, r1s2, ROWS);

// stop timer
gettimeofday(&end, 0);
seconds = end.tv_sec - begin.tv_sec;
micro = end.tv_usec - begin.tv_usec;
elapsed = seconds + micro*1e-6;

if (rank == 0) {
  printf("BATCHED\t%ld\t%.3f\n", ROWS, elapsed);
}
```

Input Arguments

Experiment Timing

After

```c
int main(int argc, char** argv) {

  if (argc < 5) {
    printf("\n\nUsage: %s [INPUT_SIZE_START] [STEP_SIZE] [NUM_STEPS] [NUM_ITER]\n\n", argv[0]);
    return -1;
  }
```

```c

      /* ======================================================
        1. Measure exchange_array
      ====================================================== */
      // start timer
      // CLOCK_PROCESS_CPUTIME_ID or CLOCK_MONOTONIC perhaps (check man clock_gettime)
      clock_gettime(CLOCK_MONOTONIC, &begin);

      exchange_shares_array(r1s1, r1s2, CURR_ROW);

      // stop timer
      clock_gettime(CLOCK_MONOTONIC, &end);
      elapsed = diff(begin,end);
      time_stamp[nstep][iter][2] = elapsed;

      if (rank == 0) {
        printf("BATCHED\t%ld\t%lld.%09ld\n", CURR_ROW, (long long)elapsed.tv_sec, elapsed.tv_nsec);
      }
```

```c
if (rank == 0) {
  // Output collected data to csv file
  time_t rawtime;
  struct tm * timeinfo;
  char fname[36];

  // YYYYMMDD_hhmmss_exp-exchange_log.csv
  time(&rawtime);
  timeinfo = localtime(&rawtime);
  strftime(fname,36,"%y%m%d_%H%M%S_exp-exchange_log.csv",timeinfo);
  FILE * logfile;
  logfile = fopen (fname, "w+");

  fprintf(logfile,"ROWS,GENSHR,SEEDS,BATCHED,SYNC,ASYNC,\n");
  for (int nstep = 0; nstep < NSTEP; nstep++){
    if (nstep == 0)
      CURR_ROW = ROWS;
    else
      CURR_ROW *= STEP;
    for (int iter = 0; iter < ITER; iter++){
      fprintf(logfile,"%ld,",CURR_ROW);
      for (int meas = 0; meas < MEAS; meas++){
        fprintf(logfile,"%lld.%09ld,", (long long)time_stamp[nstep][iter][meas].tv_sec, time_stamp[nstep]
          [iter][meas].tv_nsec);
      }
      fprintf(logfile,"\n");
    }
  }
  fclose(logfile);
  // Store system details to another file
}
```

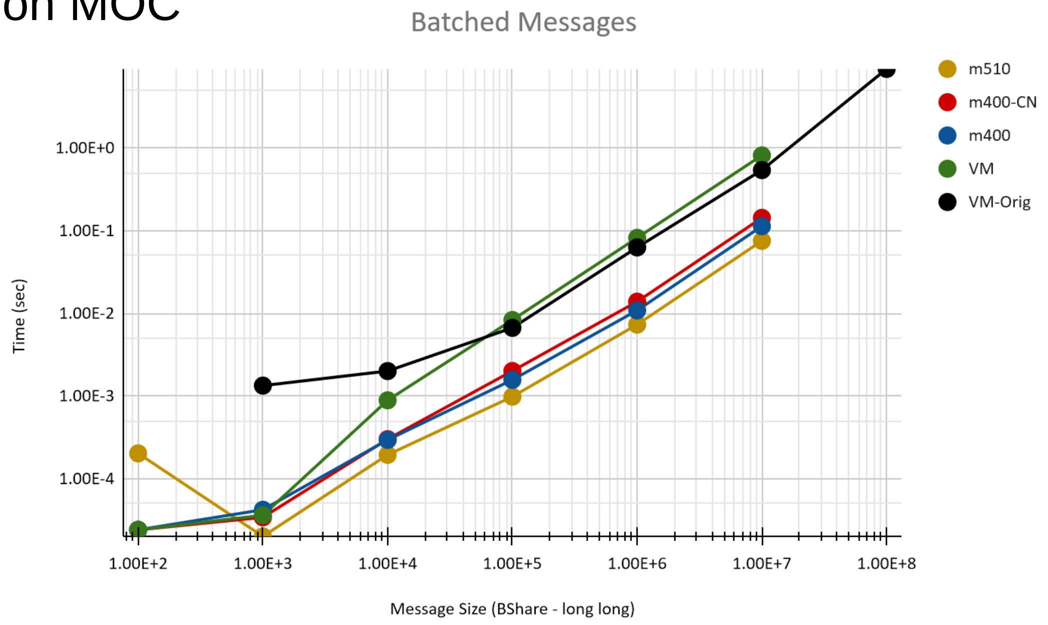
Output Format

Instrumenting the Exchange Primitive Test

- Before:
  - 3 measurements
  - gettimeofday
  - Call multiple times
    - For different sizes
    - For multiple runs
  - txt output & python import
  - No benchmark/trace collection
- After:
  - 5 measurements
  - clock_gettime
  - Call once
    - For range of sizes
    - For multiple runs
  - csv output (easier import)
  - Benchmark/trace collection with Score-P

Collected data from exp-exchange

- Overlaid batched runs from different environments
- Compare data collected from
  - Openstack on MOC
  - Cloudlab



Benchmark/Profiling

- Scorep:
  - Benchmarking/Profiling framework
  - Allows for data collection from MPI (and other sources)
  - Modify Makefile in order to build with Score-P
  - Setup overview:
    - Installed PPA package on ubuntu system
    - Install additional package (sudo apt-get install libz-dev)
    - Specify “scorep” in makefile, update all binary executable on all vms
- Analysis
  - CUBE GUI: Inspecting *.cubex benchmark file

Running Exp-exchange with Score-P

```
ylin@cc-mpc-main:~$ ls scorep-20201207_1014_3385102085294411/
MANIFEST.md  profile.cubex  scorep.cfg
```

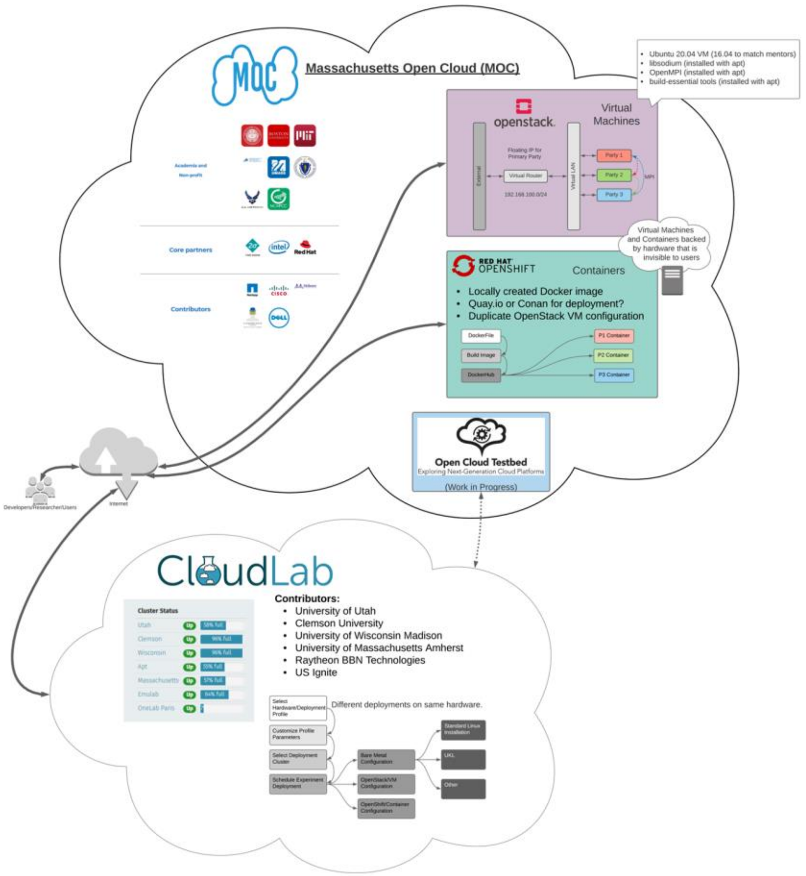
- Execute new binary file → Scorep folder including *.cubex file to evaluate → analysis of data
- Textual output supported when no Gui tool available
- Use cube to analyze data in *.cubex (details to be continued)

```
ylin@cc-mpc-main:/mpc_shared/ccproject/experiments$ cat Makefile
# CC= gcc -std=c99
# CFLAGS= -O3 -Wall -lsodium     #Use this CFLAGS when deploying in OSX
CFLAGS= -O3 -Wall                #Use this CFLAGS & DEP when deploying in
DEP= -lsodium -lm                #Use this along with CFLAGS for Linux de
MPI=scorep mpicc
SRC=../src
PRIMITIVES= $(SRC)/comm.c $(SRC)/party.c $(SRC)/primitives.c $(SRC)/shar
RELATIONAL= $(SRC)/relational.c
```

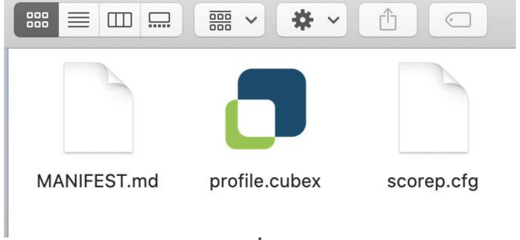
```
ylin@cc-mpc-main:~/scorep-20201207_1014_3385102085294411$ scorep-score -r profile.cubex

Estimated aggregate size of event trace:                   2432kB
Estimated requirements for largest trace buffer (max_buf): 828kB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       4097kB
(hint: When tracing set SCOREP_TOTAL_MEMORY=4097kB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)

flt     type max_buf[B] visits time[s] time[%] time/visit[us]  region
         ALL    847,365 73,111    1.41   100.0          19.28  ALL
         USR    442,494 49,055    0.01     1.0           0.28  USR
         MPI    352,700 18,038    1.38    97.8          76.47  MPI
         COM     52,130  6,015    0.02     1.2           2.76  COM
      SCOREP         41      3    0.00     0.0          21.55  SCOREP

         USR    208,234 24,027    0.00     0.2           0.13  check_init
         USR    104,130 12,015    0.00     0.2           0.25  get_rank
         MPI     89,089  3,003    0.01     0.7           3.24  MPI_Irecv
         MPI     89,089  3,003    0.07     4.9          23.13  MPI_Isend
         MPI     61,305  3,007    0.06     3.9          18.39  MPI_Send
         MPI     61,183  3,007    0.16    11.5          53.78  MPI_Recv
         USR     52,052  6,006    0.00     0.2           0.42  get_pred
         MPI     52,052  6,006    0.15    10.7          25.12  MPI_Wait
         USR     52,052  6,006    0.00     0.2           0.47  get_succ
         COM     26,000  3,000    0.01     0.6           2.69  exchange_shares_async
         COM     26,000  3,000    0.01     0.4           2.07  exchange_shares
         USR     26,000  1,000    0.00     0.2           2.24  generate_bool_share
      SCOREP         41      3    0.00     0.0          21.55  exp-exchange
         USR         26      1    0.00     0.0          59.69  init_sharing
         COM         26      3    0.00     0.0          54.74  exchange_rsz_seeds
         COM         26      3    0.00     0.0          56.31  generate_and_share_random_data
         COM         26      3    0.00     0.0          30.60  init
         MPI         26      3    0.00     0.0          14.96  MPI_Comm_rank
         MPI         26      3    0.00     0.0           1.59  MPI_Comm_size
         COM         26      3    0.00     0.0          25.63  exchange_shares_array
         MPI         26      3    0.00     0.1         268.18  MPI_Finalize
         MPI         26      3    0.93    66.1      310484.80  MPI_Init
         COM         26      3    0.00     0.1         603.82  main
```

Presentation Outline

- Overview
- Deployment Environments
- Testing Instrumentation
- Data Analysis
- Automation/Replicability
- Conclusions & Future Work



Why **CUBE Gui**



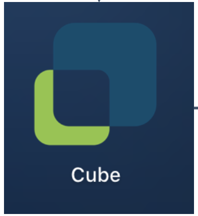
Experiment Row Data (.txt):

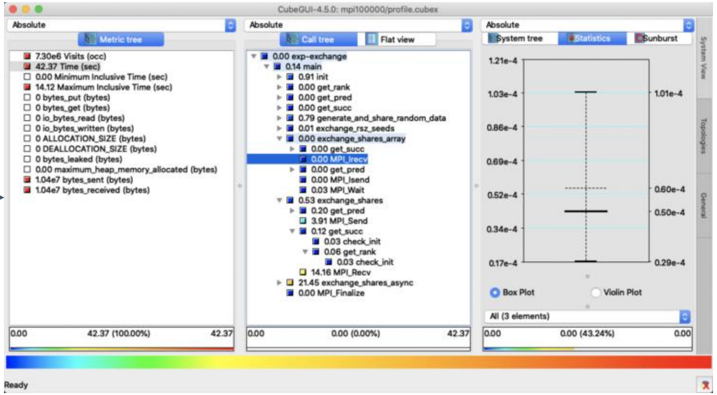
1. Limited to instrumented tests
2. No Secondary Statistic Info

Score-P Data (.cubex):

1. Inconvenient to view in text
2. No derived metrics when viewed directly

CUBE can help explore benchmark data





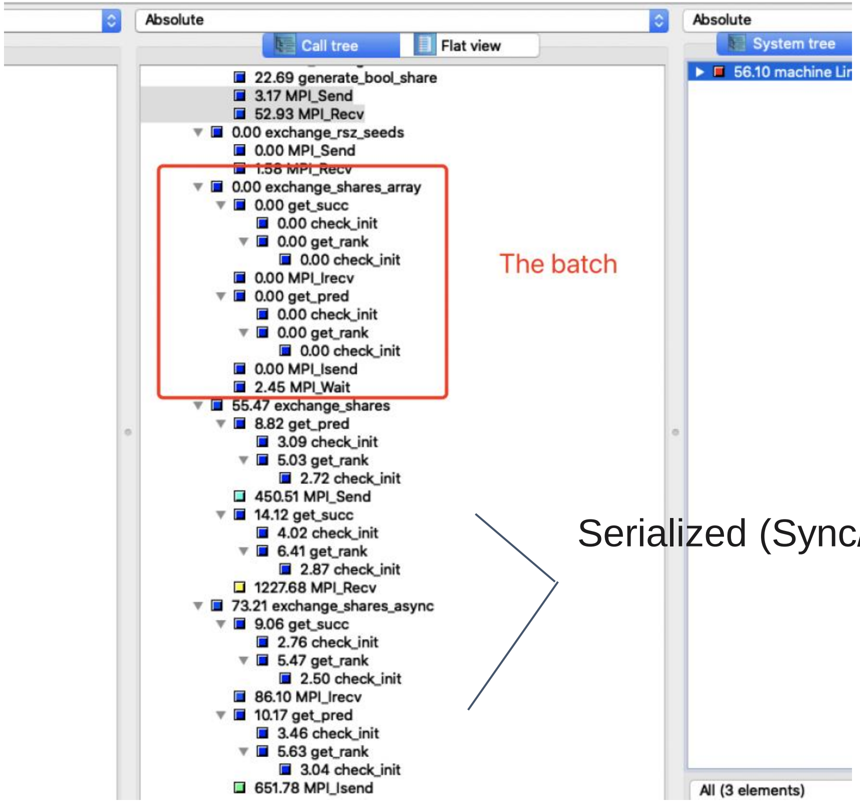
Data Insights

1. **Batch MPI** communication is **more efficient** than **Serialized (Sync/Async)**
2. Increasing batch sizes always helped in our tests
3. MPI Initialization cost is constant,

   communication efficiency reaches a steady point for larger messages

1. The standard deviation of **Sending bytes** is always **2x** of the sd of **Receiving byes.**

Batch MPI communication is **more efficient** than **Serialized (Sync/Async)**



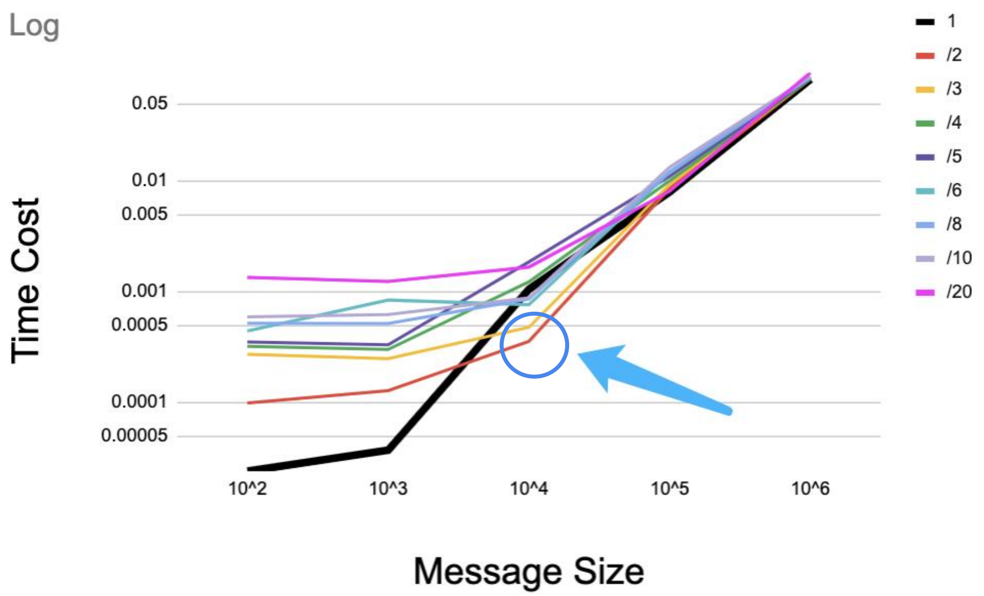
The batch

Serialized (Sync/Async)

1/100 of the time cost
for every message size

in 10^2 / 10^3 / 10^4 / 10^5 / 10^6

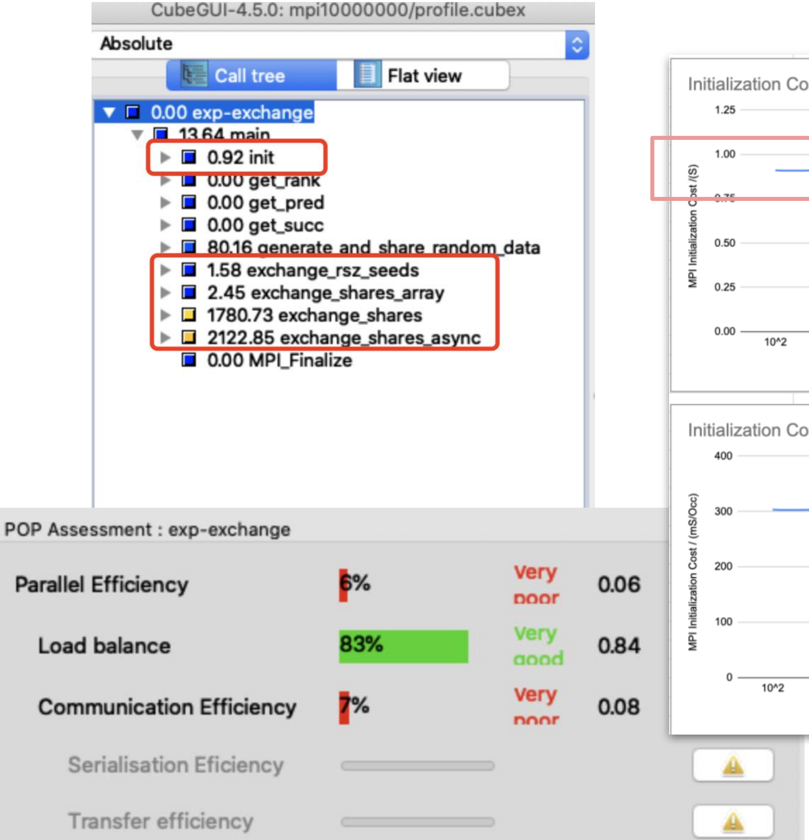
Increasing batch sizes always helped in our tests.

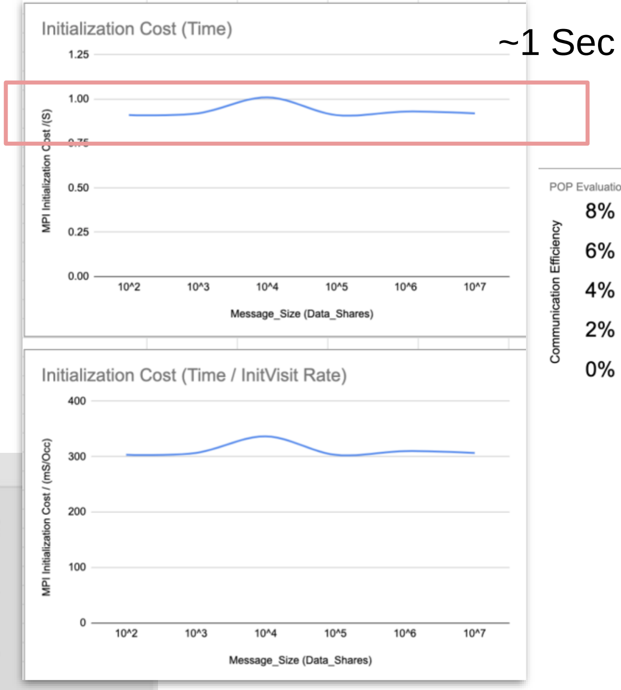


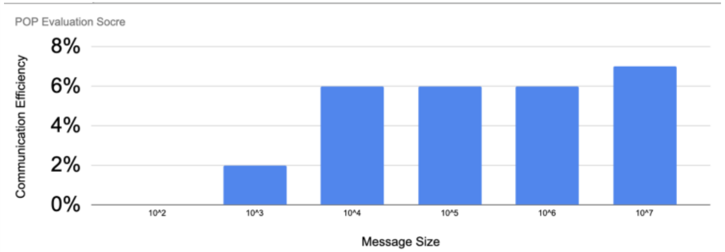

```
/* ========================================================
   1. Measure exchange_array
======================================================== */
// start timer
// CLOCK_PROCESS_CPUTIME_ID or CLOCK_MONOTONIC perhaps (check man clock_gettime)
clock_gettime(CLOCK_MONOTONIC, &begin);

long part_CURR_ROW = CURR_ROW/6;
exchange_shares_array(r1s1, r1s2, part_CURR_ROW);
exchange_shares_array(r1s1, r1s2, part_CURR_ROW);
exchange_shares_array(r1s1, r1s2, part_CURR_ROW);
exchange_shares_array(r1s1, r1s2, part_CURR_ROW);
exchange_shares_array(r1s1, r1s2, part_CURR_ROW);
exchange_shares_array(r1s1, r1s2, part_CURR_ROW);

// stop timer
clock_gettime(CLOCK_MONOTONIC, &end);
elapsed = diff(begin,end);
time_stamp[nstep][iter][2] = elapsed;

if (rank == 0) {
  printf("BATCHED\t%ld\t%lld.%09ld\n", CURR_ROW, (long long)elapsed.tv_sec, elapsed.tv_nsec);
}

/* ========================================================
   2. Measure synchronous element-wise exchange
======================================================== */
// start timer
clock_gettime(CLOCK_MONOTONIC, &begin);

for (long i=0; i<CURR_ROW; i++) {
```

Initialization cost is constant,

communication efficiency: We have low efficiency for small messages which increases with message size until hitting reaching a steady efficiency value.

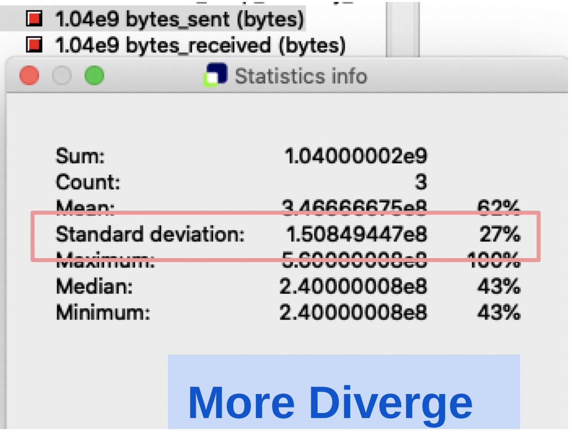




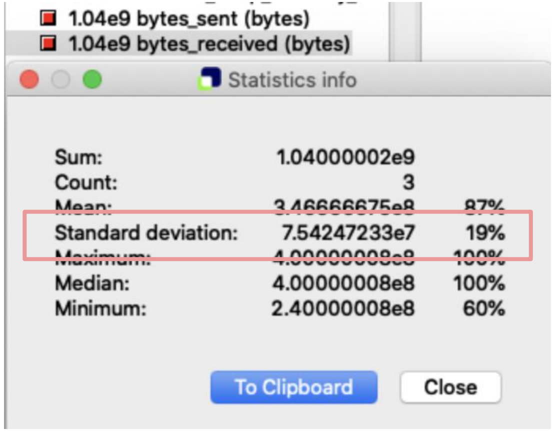


BOSTON UNIVERSITY

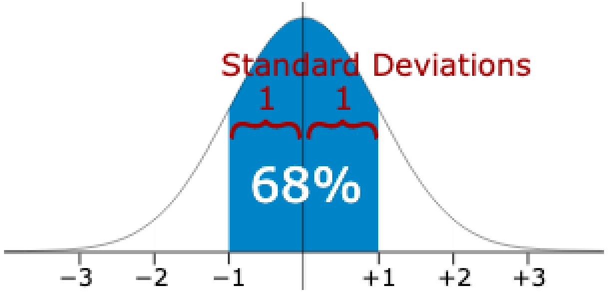
The standard deviation of **Sending bytes** is always 2x of the SD of **Receiving byes.**



Sending



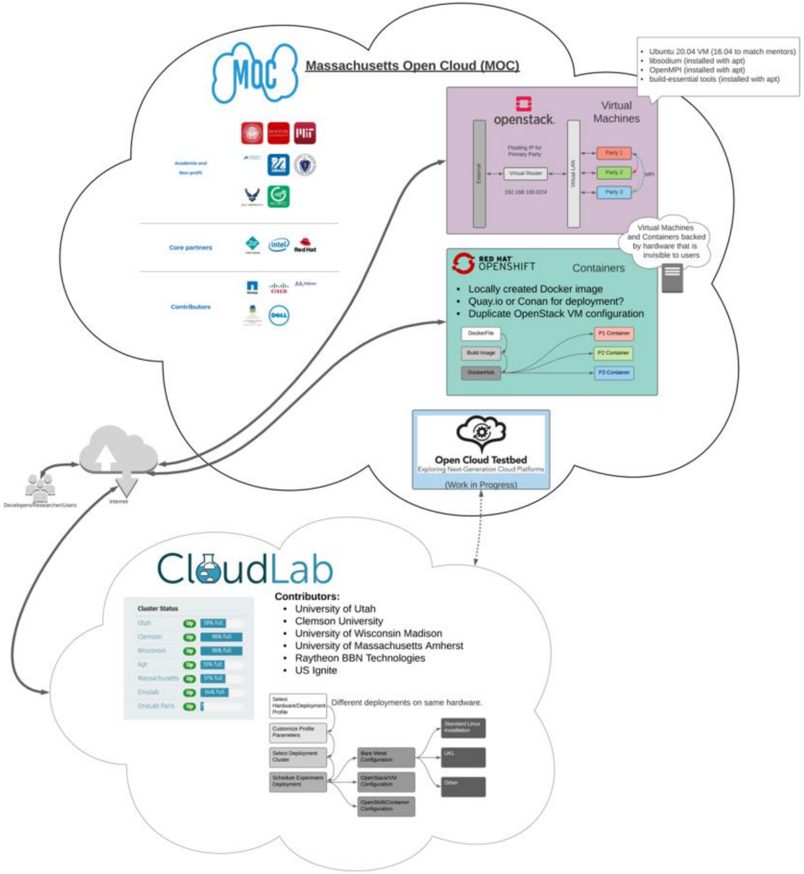
Receiving



This is interesting to note and we hypothesize that this could indicate the opportunity for further optimization in the **sending** process.
Independently testing Sync and Asyncs MPI transmissions might be a good follow-on experiment.

Presentation Outline

- Overview
- Deployment Environments
- Testing Instrumentation
- Data Analysis
- **Automation/Replicability**
- Conclusions & Future Work



Improved Automation

- Manual setup and testing
- Scripts - semi-automated
  - Shell Scripts
  - Packaged source *.tar.gz
- Ansible - advanced automation
  - MOC
  - Cloudlab

```bash
#!/bin/bash

# Installer script for CloudLab

# Check Linux Version
OS=`lsb_release -i | cut -f2`
RELEASE=`lsb_release -r | cut -f2`

# Alternate approach...use this to support CentOS eventually...
#grep ^NAME /etc/os-release

# Ubuntu
if [[ $OS == "Ubuntu" ]]; then
  # Perform update
  apt update -y && apt upgrade -y
  # Install dependencies
  if [[ $RELEASE == "16.04" ]]; then
    apt install -y make gcc libopenmpi-dev openmpi-bin libsodium18 libsodium-dev
  elif [[ $RELEASE == "18.04" ]]; then
    apt install -y make gcc libopenmpi-dev openmpi-bin libsodium23 libsodium-dev
  elif [[ $RELEASE == "20.04" ]]; then
    apt install -y make gcc libopenmpi-dev openmpi-bin libsodium23 libsodium-dev
  else
    echo "Some other Ubuntu version"
  fi

# Other
else
  echo "Not Ubuntu"
fi

# Build MPC codebase (This will only work if the dependencies were successfully installed)
cd experiments
make all

# Now we're done with setup and can run something like this:
# mpirun --host localhost,192.168.1.2,192.168.1.3 -np 3 hostname
# mpirun --host localhost,192.168.1.2,192.168.1.3 -np 3 exp-exchange 1000
```

```bash
#!/bin/bash

# Bundle MPC Source code and Installer script for CloudLab
# MPC tests and experiements on Ubuntu 20.04 setup

# -c : creates a new archive
# -z : uses gzip compression
# -f : specifies files to be added
# In this case I only want *.c, *.h, and Makefile
# I use some regex in the paths specified to achieve this
tar -czf cloudlab_setup.tar.gz ../src/*[.][ch] ../tests/{*[.][ch],Makefile} ../experiments/{*[.][ch],Makefile} setup.sh
```

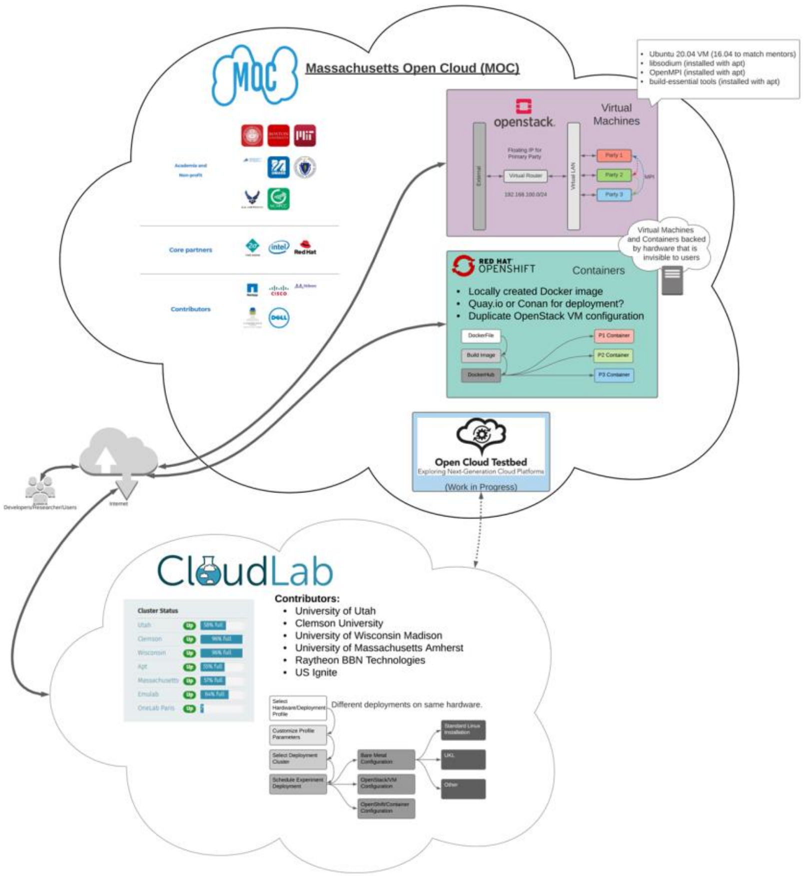

BOSTON UNIVERSITY

From a user perspective...

1. Instantiate three nodes with and OS installed.
   a. VMs on OpenStack (GUI web interface)
   b. Bare-metal nodes on CloudLab (geni-lib script)
2. Clone project git repository
3. Modify ansible environment files
   a. Specify SSH key for accessing nodes
   b. Modify IP address or name specifying nodes
4. Conditional: If using a different OS (not Ubuntu 20.04)
   a. Create a file specifying OS version of packages
5. Run ad-hoc ansible commands or playbook
   a. Playbook will execute an entire experiment scenario

Presentation Outline

- Overview
- Deployment Environments
- Testing Instrumentation
- Data Analysis
- Automation/Replicability
- Conclusions & Future Work



Conclusion - Lessons Learned

- OpenStack experience
- Docker/OpenShift experience
- Linux system administration/configuration
- MPI instrumenting and testing
- Geni-lib scripting
- Advanced Shell Scripting
- Ansible first time experience

Links to documentation and Repo here

- Github repository
  - Public: https://github.com/BU-CLOUD-F20/Secure_MultiParty
  - Private: https://github.com/jliagouris/ccproject
- Google Drive (data collection)
  - https://drive.google.com/drive/folders/143M9s_VxxjImq6VWSjr6qNxkYMCzSz9j?usp=sharing
  - Request access to other resources from project team
- Additional Resources:
  - Massachusetts Open Cloud (MOC): https://massopen.cloud/
  - Cloudlab (Bare-metal): https://www.cloudlab.us/
  - Open-MPI: https://www.open-mpi.org/
  - Score-P: https://www.vi-hps.org/projects/score-p
  - Scalasca (CUBE GUI and more…): https://www.scalasca.org/
  - Ansible: https://docs.ansible.com/ansible/latest/index.html

Future Work - Ongoing Efforts

- End-to-end application on this framework
- What is the best deployment (performance)?
  - Openstack/OpenShift on Bare-Metal (follow-on test)
- Build user-friendly front-end UI to perform steps
- More extensive data analysis for MPI using Score-P
- Deploy the codebase on Linux Unikernel

Thank you

…any questions?