Simulation of CSE Lab Cluster

CS633 2018-19-II Course Project

Yash Srivastav (150839)

March 12, 2018

Table of Contents

CSE Lab Cluster

- Consists of $\sim$ 30 PCs connected over LAN

CSE Lab Cluster

- Consists of ~ 30 PCs connected over LAN
- Hostfiles used for MPI programs usually don't take topology or placement into account

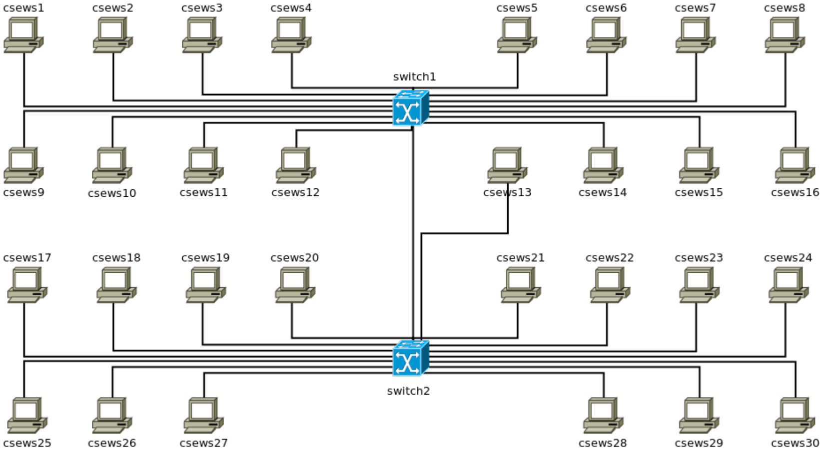
CSE Lab Cluster

- Consists of ~ 30 PCs connected over LAN
- Hostfiles used for MPI programs usually don't take topology or placement into account
- We want to simulate the running of an MPI program on this cluster and predict time taken for communications

CSE Lab Cluster

- Consists of $\sim$ 30 PCs connected over LAN
- Hostfiles used for MPI programs usually don't take topology or placement into account
- We want to simulate the running of an MPI program on this cluster and predict time taken for communications
- Would enable simulating under different hostfiles and use it to decide upon which one to use

CSE Lab Cluster Topology



- `csews13` is on different switch from remaining `csews1-16`

Table of Contents

Related Research Papers

- MPI-NeTSim: A network simulation module for MPI [PWTR09]
- Simulation of MPI Applications with Time-independent Traces [CDMS15]
- Single Node On-Line Simulation of MPI Applications with SMPI [CSG+11]
- Toward Better Simulation of MPI Applications on Ethernet/TCP Networks [BDG+14]
- Simulating MPI applications: the SMPI approach [DLM+17]

Table of Contents

Initial Approach (without referring to related work)

- Obtain event traces for the target MPI program by running it on a host machine with the required number of processes.

Initial Approach (without referring to related work)

- Obtain event traces for the target MPI program by running it on a host machine with the required number of processes.
- Separate all communications into time slots which can be simulated as occurring concurrently.

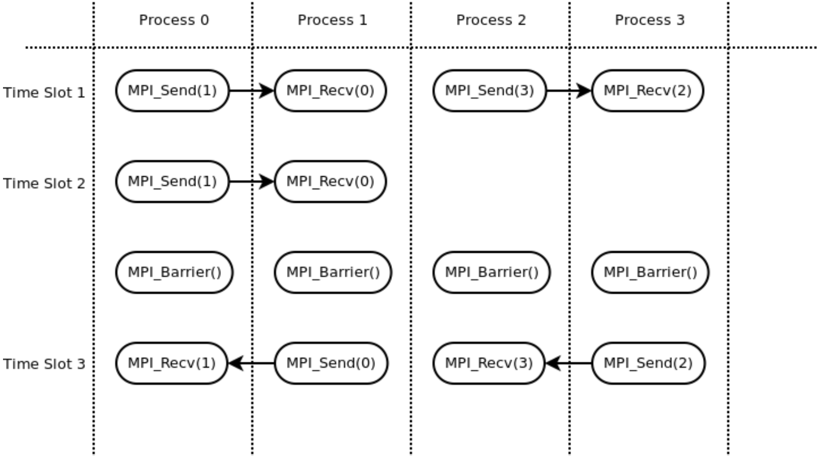
Initial Approach (without referring to related work)

- Obtain event traces for the target MPI program by running it on a host machine with the required number of processes.
- Separate all communications into time slots which can be simulated as occurring concurrently.
- Using the knowledge of the network topology and a given hostfile, perform simulation based on the trace obtained in a network simulation framework.

Initial Approach (without referring to related work)

- Obtain event traces for the target MPI program by running it on a host machine with the required number of processes.
- Separate all communications into time slots which can be simulated as occurring concurrently.
- Using the knowledge of the network topology and a given hostfile, perform simulation based on the trace obtained in a network simulation framework.
- Report the expected time taken for communications and congestion points/links, if any.

Sample Time Slotting



Drawbacks/Challenges

- Doesn't take into account the computation period in-between communications.

Drawbacks/Challenges

- Doesn't take into account the computation period in-between communications.
- Even if we were to take computation period into account, there is no reliable way of getting it on an over-subscribed host as `MPI_WTime` returns values from fixed past time and does not take process scheduling into account.

Drawbacks/Challenges

- Doesn't take into account the computation period in-between communications.
- Even if we were to take computation period into account, there is no reliable way of getting it on an over-subscribed host as `MPI_WTime` returns values from fixed past time and does not take process scheduling into account.
- Also computation time would come out to be different for different ranked processes based on scheduling / other loads on the CPU / etc even if they were running the same set of instructions.

Refined Approach (inspired by related work)

- Obtain time-independent event traces for the target MPI program similar to [CDMS15].

Refined Approach (inspired by related work)

- Obtain time-independent event traces for the target MPI program similar to [CDMS15].
- The above gives us bytes transferred (for communications) and number of instructions executed (for computations).

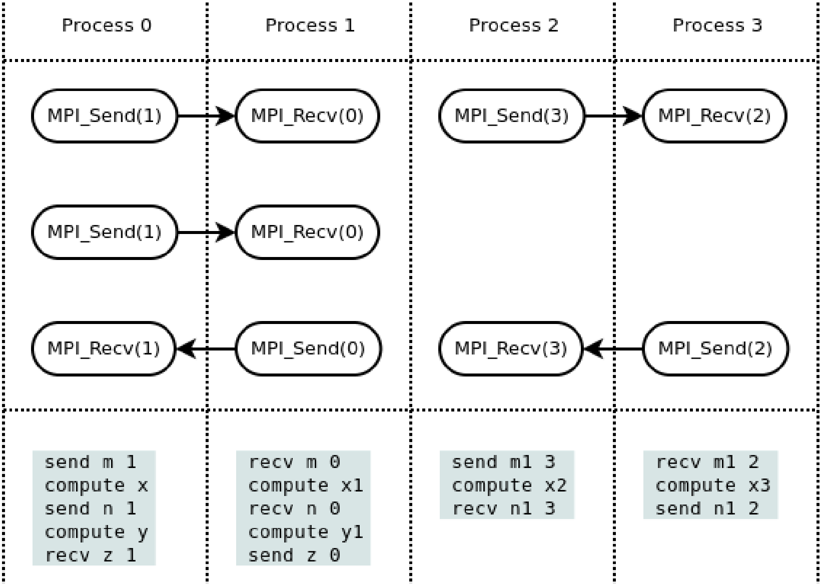
Refined Approach (inspired by related work)

- Obtain time-independent event traces for the target MPI program similar to [CDMS15].
- The above gives us bytes transferred (for communications) and number of instructions executed (for computations).
- Using the knowledge of the network topology and a given hostfile, perform simulation based on the trace obtained in an event-based network simulation framework.

Refined Approach (inspired by related work)

- Obtain time-independent event traces for the target MPI program similar to [CDMS15].
- The above gives us bytes transferred (for communications) and number of instructions executed (for computations).
- Using the knowledge of the network topology and a given hostfile, perform simulation based on the trace obtained in an event-based network simulation framework.
- Report the expected time taken for communications and congestion points/links, if any.

Sample Trace



Obtaining Event Traces

- We use the `PMPI_` interface exposed by nearly all MPI implementations inorder to override calls to `MPI_` functions with custom code. This allows us to log calls.

Obtaining Event Traces

- We use the `PMPI_` interface exposed by nearly all MPI implementations inorder to override calls to `MPI_` functions with custom code. This allows us to log calls.
- We also use the **Performance Application Programming Interface (PAPI)** [MBDH99] in order to measure CPU cycles / instructions between 2 MPI calls (computation). Since PAPI counters are thread specific, this approach is valid on over-subscribed MPI executions as well.

Obtaining Event Traces

- We use the `PMPI_` interface exposed by nearly all MPI implementations inorder to override calls to `MPI_` functions with custom code. This allows us to log calls.
- We also use the **Performance Application Programming Interface (PAPI)** [MBDH99] in order to measure CPU cycles / instructions between 2 MPI calls (computation). Since PAPI counters are thread specific, this approach is valid on over-subscribed MPI executions as well.
- At the end of an invocation of an MPI call, we note down the value of the performance counter. At the start of the next MPI call, the difference between the counter value is the computation.

Obtaining Event Traces - Challenges / Drawbacks

- In order to collect the traces on a single machine, the machine should have enough memory for all the processes.

Obtaining Event Traces - Challenges / Drawbacks

- In order to collect the traces on a single machine, the machine should have enough memory for all the processes.
- The trace size grows pretty large for non-trivial applications.

Obtaining Event Traces - Challenges / Drawbacks

- In order to collect the traces on a single machine, the machine should have enough memory for all the processes.
- The trace size grows pretty large for non-trivial applications.
- This would also log internal `MPI_Send` and `MPI_Recv` used by MPI Collectives. For that, we could set a flag before a collective call and reset it after the call.

Obtaining Event Traces - Challenges / Drawbacks

- In order to collect the traces on a single machine, the machine should have enough memory for all the processes.
- The trace size grows pretty large for non-trivial applications.
- This would also log internal `MPI_Send` and `MPI_Recv` used by MPI Collectives. For that, we could set a flag before a collective call and reset it after the call.
- The tracing machine should have the same processor architecture as the CSE lab cluster. Since our tracer would run on one of the lab machines, it is not an issue.

Obtaining Event Traces - Challenges / Drawbacks

- In order to collect the traces on a single machine, the machine should have enough memory for all the processes.
- The trace size grows pretty large for non-trivial applications.
- This would also log internal `MPI_Send` and `MPI_Recv` used by MPI Collectives. For that, we could set a flag before a collective call and reset it after the call.
- The tracing machine should have the same processor architecture as the CSE lab cluster. Since our tracer would run on one of the lab machines, it is not an issue.
- The computation time during MPI calls is ignored by our simulation but its ok because its expected for it be negligible compared to the communication time over the network.

Offline Simulation

- Using the information collected by the traces and a provided hostfile, we could generate an event stream for each node on the cluster.

Offline Simulation

- Using the information collected by the traces and a provided hostfile, we could generate an event stream for each node on the cluster.
- The above event stream would be fed into an event-based network simulator like ns3 which has all the hosts configured as per the cse lab cluster topology.

Offline Simulation

- Using the information collected by the traces and a provided hostfile, we could generate an event stream for each node on the cluster.
- The above event stream would be fed into an event-based network simulator like ns3 which has all the hosts configured as per the cse lab cluster topology.
- This should give us the expected time for each event and also allow us to identify limiting link contentions.

Offline Simulation

- Using the information collected by the traces and a provided hostfile, we could generate an event stream for each node on the cluster.
- The above event stream would be fed into an event-based network simulator like ns3 which has all the hosts configured as per the cse lab cluster topology.
- This should give us the expected time for each event and also allow us to identify limiting link contentions.
- **NOTE:** The ns3 framework needs a little more exploration to finalize this part.

Table of Contents

MPI Calls

The final simulator should be able to simulate/handle the following synchronous MPI calls with accurate algorithms as used in default `MPICH v3.2.1`:

- `MPI_Send`
- `MPI_Recv`
- `MPI_Broadcast`
- `MPI_Reduce`
- `MPI_Scatter`
- `MPI_Allreduce`
- `MPI_Alltoall`
- `MPI_Gather`
- `MPI_Allgather`
- `MPI_Barrier`

Simulation Semantics

This is how the final **Sim**ulated **MPI** should be invoked:

```
$ simpicc $FLAGS -o prog.x prog.c
$ simpirun -f hostfile -np 32 prog.x
```

`hostfile` should contain only hosts on the cse lab cluster with hostnames as `csews1`, etc.

Table of Contents

MPI Calls

The final simulator may be able to simulate/handle the following MPI calls:

- Async variants and associated calls like `MPI_Wait`
- Vector variants (like `MPI_Allgatherv`)

Simulation Semantics

This is how the final **Sim**ulated **MPI** may be invoked (I'm not sure if this is possible):

```
$ export LD_PRELOAD=/path/to/simpilib/
$ simpirun -f hostfile -np 32 prog.x
```

This would allow us to simulate precompiled binaries without modifying the makefile or recompiling.

Misc Targets

- Optimized tracefile size by using better binary formats like protobuf for trace storage.

Misc Targets

- Optimized tracefile size by using better binary formats like protobuf for trace storage.
- Proper `autotools` based installation setup for `simpi`.

Table of Contents

Future Possibilities

Apart from the maybes described earlier, there are quite a few improvements which could be made:

- Distributed trace collection: The trace collection itself could be run parallelly on multiple machines for faster trace collection and simulation.
- Including / Predicting cpu cycles for the communication algorithm itself.
- Packet level simulation which is harder to get right.

Trace Generation

- The current implementation of libsimpi can be used to generate traces for `MPI_Send` and `MPI_Recv`.

Trace Generation

- The current implementation of libsimpi can be used to generate traces for `MPI_Send` and `MPI_Recv`.
- Extending to others should not be difficult.

Trace Generation

- The current implementation of libsimpi can be used to generate traces for `MPI_Send` and `MPI_Recv`.
- Extending to others should not be difficult.
- Next target is to obtain computation count/cycles.

Sample MPI_Send PMPI callback

```
int MPI_Send(const void *buffer, int count,
             MPI_Datatype datatype, int dest,
             int tag, MPI_Comm comm) {
  int size;
  int result = PMPI_Send(buffer, count,
                   datatype, dest, tag, comm);
  PMPI_Type_size(datatype, &size);
  fprintf(stderr, "[%d] send %d to %d\n",
    rank, count * size, dest);
  return result;
}
```

Sample Trace

```
$ mpicc -g -O3 -profile=simpi asgn1-1.o -o asgn1-1.x  # li
$ mpirun -np 16 ./asgn1-1.x 1 > /dev/null  # discard stdout
[0] send 1024 to 1
[1] recv 1024 from 0
[2] send 1024 to 3
[3] recv 1024 from 2
[4] send 1024 to 5
[5] recv 1024 from 4
[6] send 1024 to 7
[7] recv 1024 from 6
[8] send 1024 to 9
[9] recv 1024 from 8
[10] send 1024 to 11
[11] recv 1024 from 10
[12] send 1024 to 13
[14] send 1024 to 15
[13] recv 1024 from 12
```

Thank You!

"So long and thanks for all the fish." – Douglas Adams

Table of Contents

References I

Paul Bédaride, Augustin Degomme, Stéphane Genaud, Arnaud Legrand, George S. Markomanolis, Martin Quinson, Mark Stillwell, Frédéric Suter, and Brice Videau, *Toward better simulation of mpi applications on ethernet/tcp networks*, High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation (Cham) (Stephen A. Jarvis, Steven A. Wright, and Simon D. Hammond, eds.), Springer International Publishing, 2014, pp. 158–181.

Henri Casanova, Frédéric Desprez, George S. Markomanolis, and Frédéric Suter, *Simulation of mpi applications with time-independent traces*, Concurr. Comput. : Pract. Exper. **27** (2015), no. 5, 1145–1168.

References II

- P. Clauss, M. Stillwell, S. Genaud, F. Suter, H. Casanova, and M. Quinson, *Single node on-line simulation of mpi applications with smpi*, 2011 IEEE International Parallel Distributed Processing Symposium, May 2011, pp. 664–675.
- Augustin Degomme, Arnaud Legrand, Georges Markomanolis, Martin Quinson, Mark Lee Stillwell, and Frédéric Suter, *Simulating MPI applications: the SMPI approach*, IEEE Transactions on Parallel and Distributed Systems **28** (2017), no. 8, 14.
- Philip J. Mucci, Shirley Browne, Christine Deane, and George Ho, *Papi: A portable interface to hardware performance counters*, In Proceedings of the Department of Defense HPCMP Users Group Conference, 1999, pp. 7–10.

References III

B. Penoff, A. Wagner, M. Tüxen, and I. Rüngeler, *Mpi-netsim: A network simulation module for mpi*, 2009 15th International Conference on Parallel and Distributed Systems, Dec 2009, pp. 464–471.