Simulation of CSE Lab Cluster

CS633 2018-19-II Course Project

Yash Srivastav (150839)

April 25, 2018

Table of Contents

CSE Lab Cluster

- Consists of ~ 30 PCs connected over LAN

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

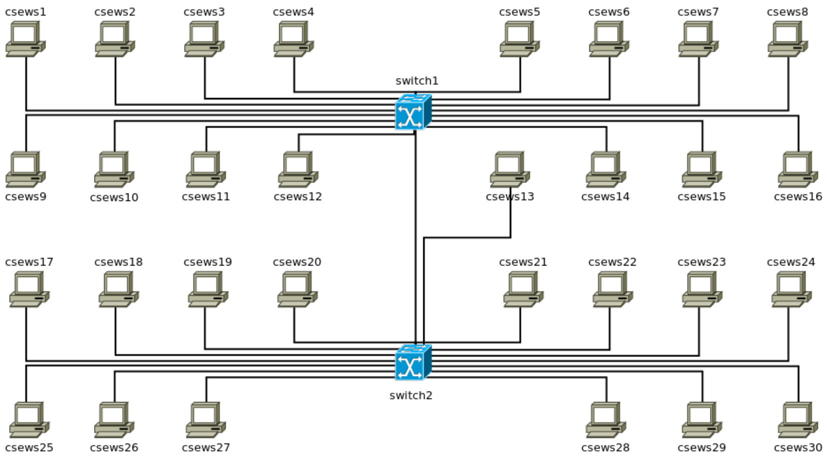
CSE Lab Cluster

- 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 ~ 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 [1]
- Simulation of MPI Applications with Time-independent Traces [2]
- Single Node On-Line Simulation of MPI Applications with SMPI [3]
- Toward Better Simulation of MPI Applications on Ethernet/TCP Networks [4]
- Simulating MPI applications: the SMPI approach [5]

Possible Approaches

- Offline (post-mortem): Collect a trace of all functions being called and their arguments during a sample run. Use this trace to simulate any target run.

Possible Approaches

- Offline (post-mortem): Collect a trace of all functions being called and their arguments during a sample run. Use this trace to simulate any target run.
- Online (in-situ): Perform simulation while executing the program itself, usually by hooking into the network layer and redirecting network requests to a simulator.

Possible Approaches

- Offline (post-mortem): Collect a trace of all functions being called and their arguments during a sample run. Use this trace to simulate any target run.
- Online (in-situ): Perform simulation while executing the program itself, usually by hooking into the network layer and redirecting network requests to a simulator.
- Hybrid: A combination of the above to have prior knowledge of the types of network communications supposed to happen and simulate better accordingly.

Salient points of previous papers

- [1] uses an online approach which hooks onto the MPI network layer and simulates it. Due to latency of simulation, the normal code itself has to be scaled to slow down. This may lead to slower simulations

Salient points of previous papers

- [1] uses an online approach which hooks onto the MPI network layer and simulates it. Due to latency of simulation, the normal code itself has to be scaled to slow down. This may lead to slower simulations
- [5] and [3] use a complete reimplementation of the MPI specification on the **SimGrid** simulation kernek with a hybrid approach. Hence its not possible to switch out one MPI implementation for another.

Salient points of previous papers

- [1] uses an online approach which hooks onto the MPI network layer and simulates it. Due to latency of simulation, the normal code itself has to be scaled to slow down. This may lead to slower simulations
- [5] and [3] use a complete reimplementation of the MPI specification on the **SimGrid** simulation kernek with a hybrid approach. Hence its not possible to switch out one MPI implementation for another.
- Even [2] uses the above smpi implementation even though it is an offline approach

Table of Contents

Approach

- Obtain time-independent event traces for the target MPI program similar to [2].

Approach

- Obtain time-independent event traces for the target MPI program similar to [2].
- The above gives us bytes transferred and to whom (for communications) and number of instructions executed (for computations).

Approach

- Obtain time-independent event traces for the target MPI program similar to [2].
- The above gives us bytes transferred and to whom (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 on **ns3**.

Approach

- Obtain time-independent event traces for the target MPI program similar to [2].
- The above gives us bytes transferred and to whom (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 on **ns3**.
- Report the expected time taken.

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 trace calls to those functions.

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 trace calls to those functions.
- We also use the **Performance Application Programming Interface (PAPI)** [6] 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 trace calls to those functions.
- We also use the **Performance Application Programming Interface (PAPI)** [6] 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 are the computation instructions.

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.
- 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.

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.

Table of Contents

MPI Calls

The simulator simulates/handles the following synchronous MPI calls with algorithms as used in default `MPICH v3.2.1`:

- `MPI_Send`
- `MPI_Recv`
- `MPI_Scatter`
- `MPI_Gather`
- `MPI_Broadcast` (only the algorithm for short messages for now)

Simulation Semantics (tracing)

This is how the **Sim**ulated **MPI** tracer is used:

```
$ export LD_PRELOAD=libpapi.so:libsimpi.so
$ mpirun -np 32 prog.x
$ cat *.log
```

This gives us a trace/log of all MPI function calls we are interested in.

Simulation Semantics (simulation)

This is how the simulator is invoked:

```
$ ./run-simulator.sh 64 hostfile path/to/log
```

This gives us a value for the expected simulation time.

Table of Contents

Tracer

- A compiled static library which can be used with `LD_PRELOAD`

Tracer

- A compiled static library which can be used with `LD_PRELOAD`
- Logs all relevant info of an MPI function call whenever it is encountered.

Tracer

- A compiled static library which can be used with `LD_PRELOAD`
- Logs all relevant info of an MPI function call whenever it is encountered.
- Each MPI process opens up a log file during `MPI_Init` and closes it during `MPI_Finalize`.

Simulator

- Consists of 3 major components: the parser, topology generator and the main `MPINode` `ns3::Application`.

Simulator

- Consists of 3 major components: the parser, topology generator and the main `MPINode` `ns3::Application`.
- The parser parses the given log files to generate event log for every process

Simulator

- Consists of 3 major components: the parser, topology generator and the main `MPINode` `ns3::Application`.
- The parser parses the given log files to generate event log for every process
- The topology generator sets up the entire network as per the CSE lab cluster.

Simulator

- Consists of 3 major components: the parser, topology generator and the main `MPINode` `ns3::Application`.
- The parser parses the given log files to generate event log for every process
- The topology generator sets up the entire network as per the CSE lab cluster.
- `MPINode` is an `ns3::Application` supposed to behave like our MPI process. An `ns3::Application` can be installed on any node on the network. We install 1 MPI application corresponding to each MPI process. So there are PPN number of applications running on a node.

Parser

- Parses event logs to an event stream which can be simulated by `MPINode`.

Parser

- Parses event logs to an event stream which can be simulated by `MPINode`.
- Event stream consists of only the following types of events: `Compute`, `Recv` & `Send`.

Parser

- Parses event logs to an event stream which can be simulated by `MPINode`.
- Event stream consists of only the following types of events: `Compute`, `Recv` & `Send`.
- Any other kind of event like `MPI_Scatter` is converted into a mixture of the above events as per the communications taking place according to the algorithms.

MPINode

- Essentially a tcp server + client.

MPINode

- Essentially a tcp server + client.
- Events are processed sequentially as soon as the previous one finishes. Once there are no more events left, the application is stopped. Once all applications stop we can claim that the simulation has ended.

MPINode

- Essentially a tcp server + client.
- Events are processed sequentially as soon as the previous one finishes. Once there are no more events left, the application is stopped. Once all applications stop we can claim that the simulation has ended.
- For compute events, an artificial delay is introduced.

MPINode

- Essentially a tcp server + client.
- Events are processed sequentially as soon as the previous one finishes. Once there are no more events left, the application is stopped. Once all applications stop we can claim that the simulation has ended.
- For compute events, an artificial delay is introduced.
- For send and receive, normal socket operations are used.

Table of Contents

Demo

Table of Contents

Future Possibilities

- Add support for different communicators.

Future Possibilities

- Add support for different communicators.
- Add support for handling `MPI` function errors during tracing and logging them as well for use during simulation.

Future Possibilities

- Add support for different communicators.
- Add support for handling `MPI` function errors during tracing and logging them as well for use during simulation.
- Remaining collectives and Async variants of `MPI` functions and associated calls like `MPI_Wait`

Future Possibilities

- Add support for different communicators.
- Add support for handling `MPI` function errors during tracing and logging them as well for use during simulation.
- Remaining collectives and Async variants of `MPI` functions and associated calls like `MPI_Wait`
- Vector variants of `MPI` collectives (like `MPI_Allgatherv`)

Future Possibilities

- Add support for different communicators.
- Add support for handling `MPI` function errors during tracing and logging them as well for use during simulation.
- Remaining collectives and Async variants of `MPI` functions and associated calls like `MPI_Wait`
- Vector variants of `MPI` collectives (like `MPI_Allgatherv`)
- Allow greater send and receive sizes (fix socket + packet sending issue).

Future Possibilities

- Add support for different communicators.
- Add support for handling `MPI` function errors during tracing and logging them as well for use during simulation.
- Remaining collectives and Async variants of `MPI` functions and associated calls like `MPI_Wait`
- Vector variants of `MPI` collectives (like `MPI_Allgatherv`)
- Allow greater send and receive sizes (fix socket + packet sending issue).
- Comparison with other `MPI` simulation frameworks developed as part of other works.

Future Possibilities

- Add support for different communicators.
- Add support for handling `MPI` function errors during tracing and logging them as well for use during simulation.
- Remaining collectives and Async variants of `MPI` functions and associated calls like `MPI_Wait`
- Vector variants of `MPI` collectives (like `MPI_Allgatherv`)
- Allow greater send and receive sizes (fix socket + packet sending issue).
- Comparison with other `MPI` simulation frameworks developed as part of other works.
- Distributed trace collection: The trace collection itself could be run parallelly on multiple machines for faster trace collection and simulation.

Thank You!

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

Table of Contents

References I

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

H. Casanova, F. Desprez, G. S. Markomanolis, and F. Suter, "Simulation of mpi applications with time-independent traces," *Concurr. Comput. : Pract. Exper.*, vol. 27, no. 5, pp. 1145–1168, Apr. 2015. [Online]. Available: http://dx.doi.org/10.1002/cpe.3278

P. Clauss, M. Stillwell, S. Genaud, F. Suter, H. Casanova, and M. Quinson, "Single node on-line simulation of mpi applications with smpi," in *2011 IEEE International Parallel Distributed Processing Symposium*, May 2011, pp. 664–675.

References II

P. Bédaride, A. Degomme, S. Genaud, A. Legrand, G. S. Markomanolis, M. Quinson, M. Stillwell, F. Suter, and B. Videau, "Toward better simulation of mpi applications on ethernet/tcp networks," in *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation*, S. A. Jarvis, S. A. Wright, and S. D. Hammond, Eds. Cham: Springer International Publishing, 2014, pp. 158–181.

A. Degomme, A. Legrand, G. Markomanolis, M. Quinson, M. L. Stillwell, and F. Suter, "Simulating MPI applications: the SMPI approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, p. 14, Aug. 2017. [Online]. Available: https://hal.inria.fr/hal-01415484

References III

P. J. Mucci, S. Browne, C. Deane, and G. Ho, “Papi: A portable interface to hardware performance counters,” in *In Proceedings of the Department of Defense HPCMP Users Group Conference*, 1999, pp. 7–10.