Contention-aware Performance Prediction of Parallel Programs

Pranjal Singh

Guide: Preeti Malakar

Parallel Programs

- Conventional single-core processors have hit the frequency wall and the power wall
- Parallel computing is now ubiquitous

Parallel Programs

- Conventional single-core processors have hit the frequency wall and the power wall
- Parallel computing is now ubiquitous
- Approaches to parallel computation:
  - Shared memory (needs explicit synchronisation)
  - Message passing (needs explicit existence)

Parallel Programs

- Conventional single-core processors have hit the frequency wall and the power wall
- Parallel computing is now ubiquitous
- Approaches to parallel computation:
  - Shared memory (needs explicit synchronisation)
  - Message passing (needs explicit existence)
- Challenges - Latency, Overhead, Scalability, Predictability, Portability

Message Passing Interface

- Portable message passing standard
- Widely used on supercomputers/clusters

Message Passing Interface

- Portable message passing standard
- Widely used on supercomputers/clusters
- Several open-source and proprietary implementations - MPICH (ANL), OpenMPI, IntelMPI, MS-MPI...

Message Passing Interface

- Portable message passing standard
- Widely used on supercomputers/clusters
- Several open-source and proprietary implementations - MPICH (ANL), OpenMPI, IntelMPI, MS-MPI...
- Builds on point-to-point messages

Message Passing Interface

- Processes are grouped into communicators
- Each process is assigned a rank $\in \{0, 1, \ldots N-1\}$

Message Passing Interface

- Processes are grouped into communicators
- Each process is assigned a rank $\in \{0, 1, \ldots N-1\}$
- Point-to-point send/receives

Message Passing Interface

- Processes are grouped into communicators
- Each process is assigned a rank $\in \{0, 1, \ldots N-1\}$
- Point-to-point send/receives
- Collective operations:
  - Broadcast - send to all
  - Scatter - distribute to all
  - Gather - collect messages at root
  - Reduce - find min/sum/logical AND of each member's output
  - ...

Performance Prediction

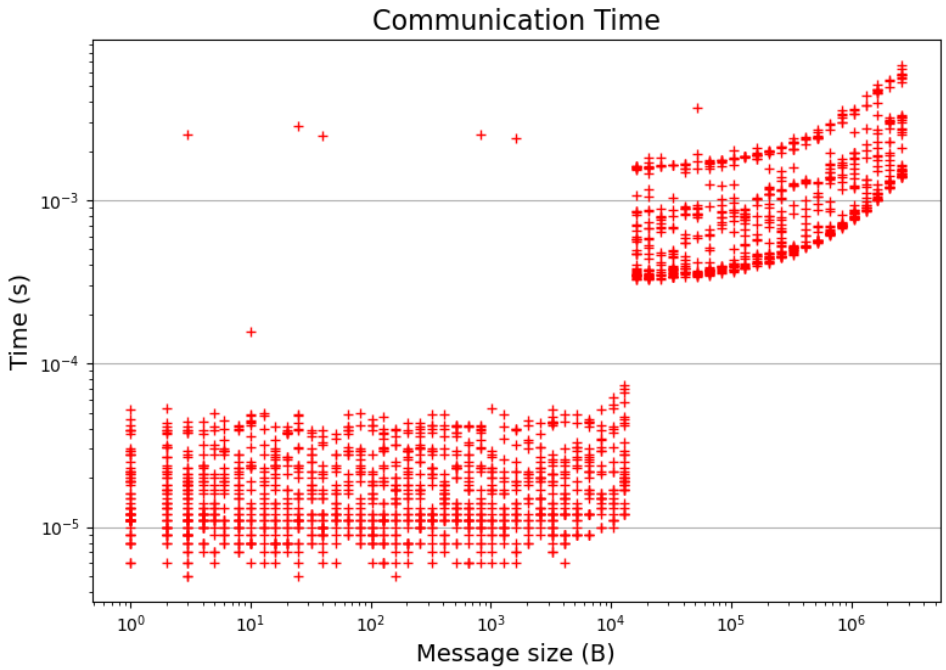
- Parallel programs use *shared* resources - network or memory
- Noise is not under control, nor predictable

Performance Prediction

- Parallel programs use *shared* resources - network or memory
- Noise is not under control, nor predictable
- Estimates needed for job scheduling

Performance Prediction

- Parallel programs use *shared* resources - network or memory
- Noise is not under control, nor predictable
- Estimates needed for job scheduling



The Hockney/Postal Model

- Splits message passing into latency and actual communication
- Mathematical expression:

$$t(n) = \alpha + n \cdot \beta = \alpha + \frac{n}{B}$$

The Hockney/Postal Model

- Splits message passing into latency and actual communication
- Mathematical expression:

$$t(n) = \alpha + n \cdot \beta = \alpha + \frac{n}{B}$$

n - message size (Bytes)
α - latency
β - reciprocal bandwidth, seconds per byte
B - Bandwidth

The Hockney/Postal Model

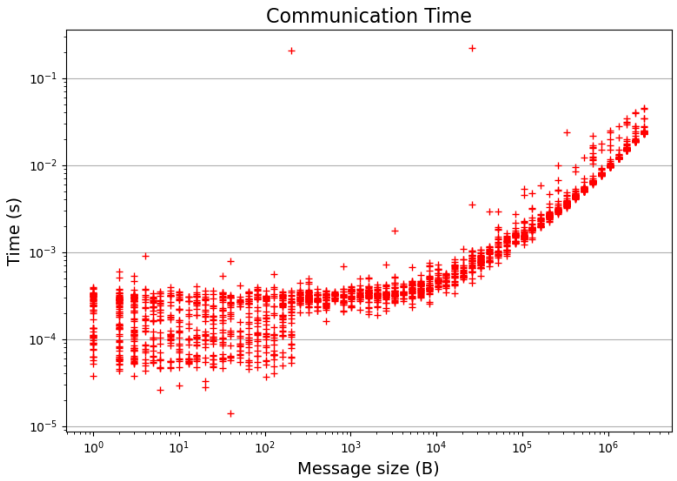
- Splits message passing into latency and actual communication
- Mathematical expression:

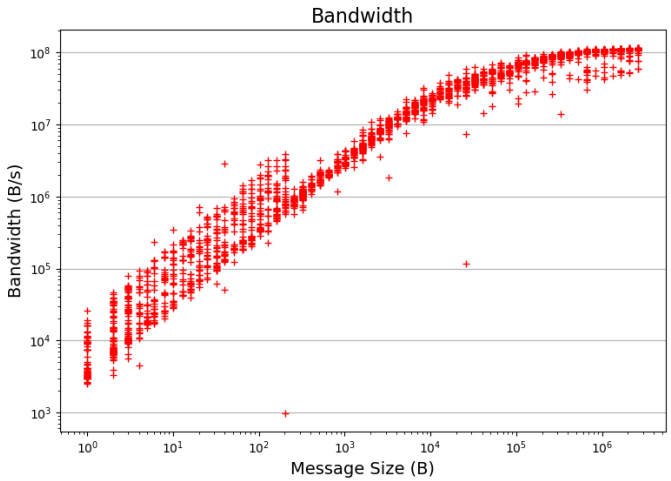
$$t(n) = \alpha + n \cdot \beta = \alpha + \frac{n}{B}$$

n - message size (Bytes)
α - latency
β - reciprocal bandwidth, seconds per byte
B - Bandwidth

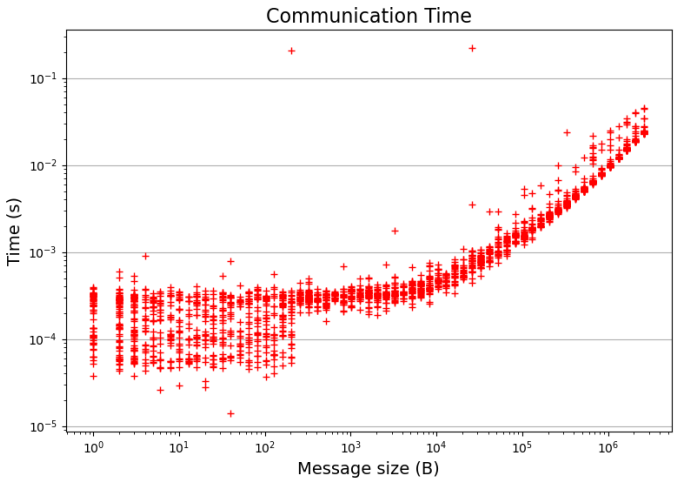
- Fairly accurate, but there are better models
- Sometimes as accurate as noise allows

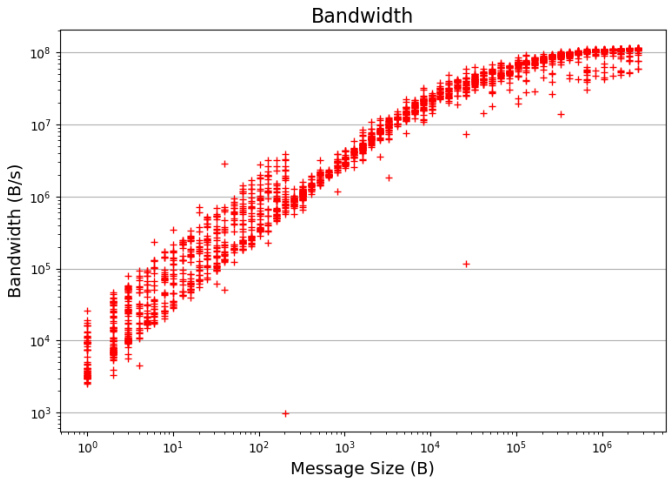
Internode Messages





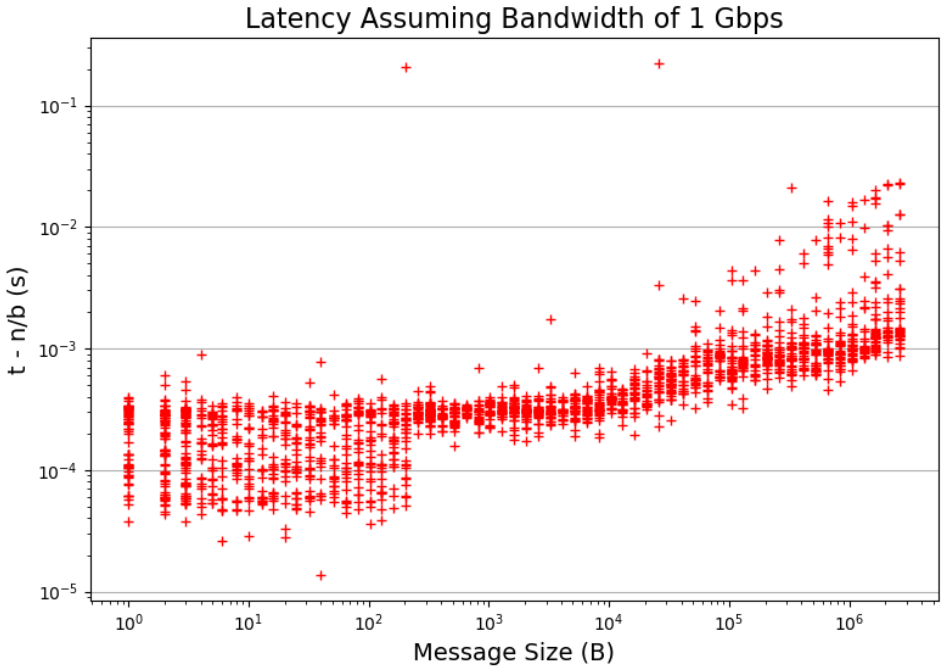
Internode Messages



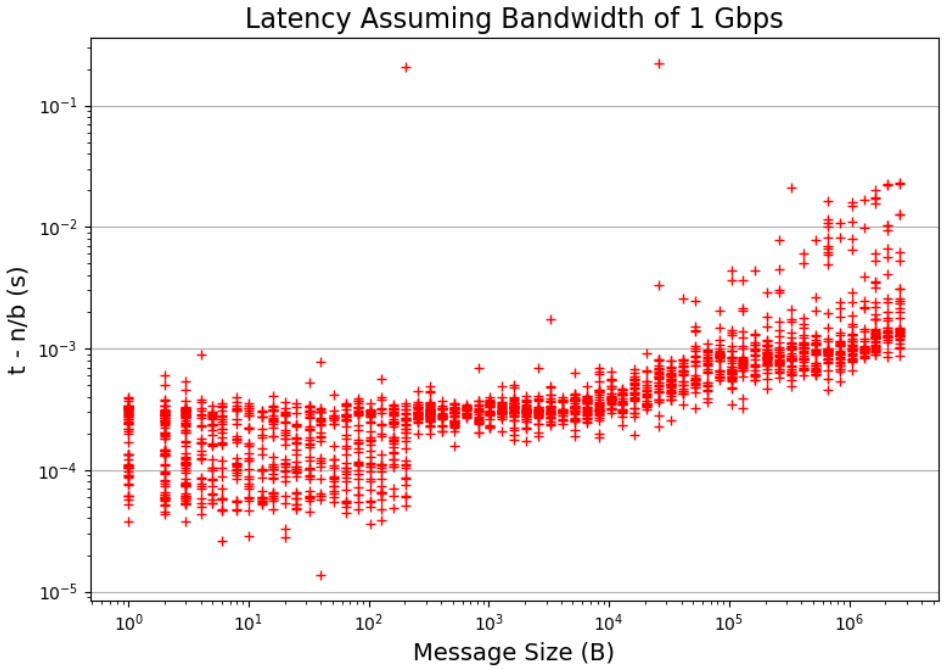


- Messages timed on CSEWS cluster - tree topology, 1 Gbps switches
- Bandwidth converges to 1 Gbps for large messages

Internode Messages

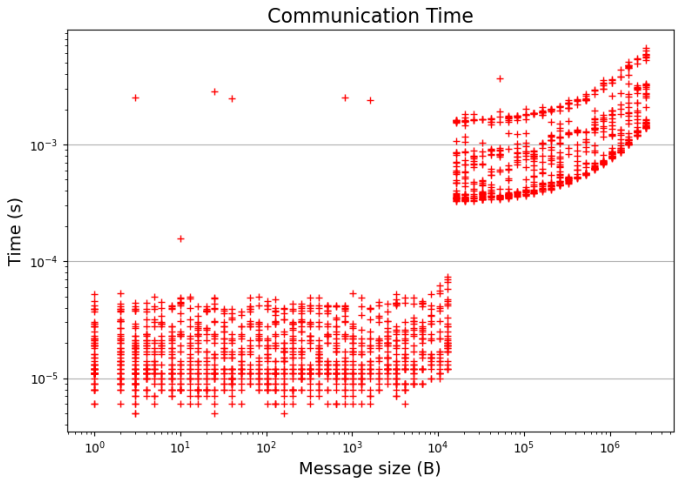


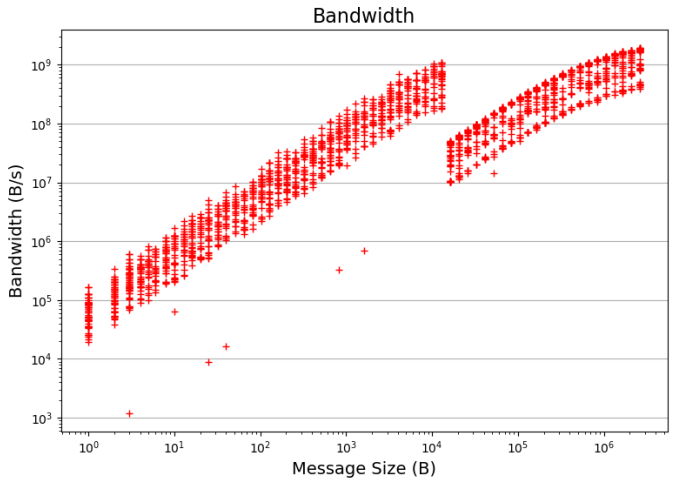
Internode Messages



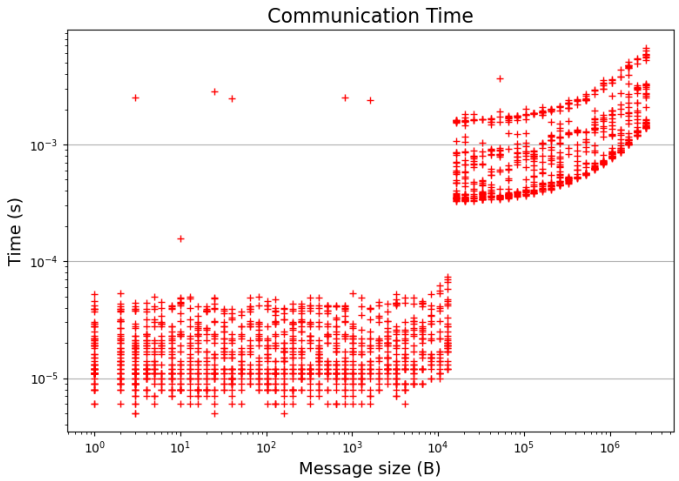
- Latency does not blow up for large messages
- Note the increase in 10 KB - 100 KB range

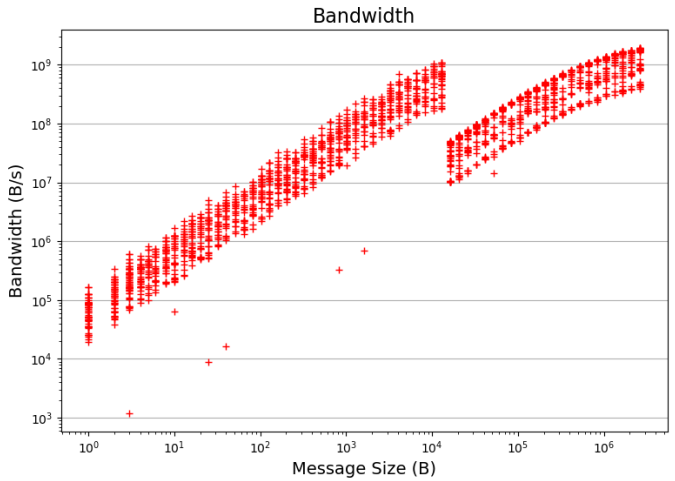
Intranode Messages





Intranode Messages





- Altogether different implementation
- Faster by orders of magnitude, not the bottleneck

Variations of the Hockney Model

- Multiple send modes - Blocking, Non-blocking, Synchronous, Ready, etc
- Internally, different modes - short/eager/rendezvous

Variations of the Hockney Model

- Multiple send modes - Blocking, Non-blocking, Synchronous, Ready, etc
- Internally, different modes - short/eager/rendezvous
- Illustrative example - Kielmann et al. obtained near-full utilisation of WAN links (7×1 Mbps) on interleaving segments across different links [Kie+01]
- Message segmentation is used now
- Hardware and software run concurrently

Network Congestion

- A lot of compute time is spent waiting for communication in supercomputers/clusters - over 80% at times
- Latency and bandwidth increase dramatically when network is being used

Network Congestion

- A lot of compute time is spent waiting for communication in supercomputers/clusters - over 80% at times
- Latency and bandwidth increase dramatically when network is being used
- No simple model of contention to date
- We tried timing messages at night/morning/midsem break
- Injected artificial contention to validate models against contention

IPMPI profiler

- Multiple algorithms used for collectives

IPMPI profiler

- Multiple algorithms used for collectives
- IPMPI profiles parallel programs
- Communication matrices - minimum, maximum, sum
- Sequence of point-to-point primitives for each collective

IPMPI profiler

- Multiple algorithms used for collectives
- IPMPI profiles parallel programs
- Communication matrices - minimum, maximum, sum
- Sequence of point-to-point primitives for each collective
- We modify IPMPI to predict execution time

Strategy 1 - Using Historical Data

- Timed P2P messages between 12 nodes - sizes from 512 B to 2 MB
- 50+ messages for each size, source and destination (1L datapoints)

Strategy 1 - Using Historical Data

- Timed P2P messages between 12 nodes - sizes from 512 B to 2 MB
- 50+ messages for each size, source and destination (1L datapoints)
- No two messages sent at the same time
- Little to no contention - data collected at odd times
- Plugged median time taken in each primitive step in a collective

Strategy 1 - Using Historical Data

- Timed P2P messages between 12 nodes - sizes from 512 B to 2 MB
- 50+ messages for each size, source and destination (1L datapoints)
- No two messages sent at the same time
- Little to no contention - data collected at odd times
- Plugged median time taken in each primitive step in a collective
- Tested on Broadcast, Allgather, Alltoall

Results

- Broadcast - about 80% accuracy
- Allgather, Alltoall - Predicts 0.5 to 0.7 times the median

Results

- Broadcast - about 80% accuracy
- Allgather, Alltoall - Predicts 0.5 to 0.7 times the median
- Usually underpredicts - congestion and synchronisation
- Slow nodes - large but accurate values predicted

Results

- Broadcast - about 80% accuracy
- Allgather, Alltoall - Predicts 0.5 to 0.7 times the median
- Usually underpredicts - congestion and synchronisation
- Slow nodes - large but accurate values predicted
- Not scalable - #messages is quadratic! Supercomputer jobs run on hundreds to thousands of cores

Strategy 2 - Accounting for contention

- Key idea - tokenize communication parameters (α, β)

Strategy 2 - Accounting for contention

- Key idea - tokenize communication parameters (α, β)
- Measure parameters for each node rather than each pair
- Sends/receives messages at each node and generates CSVs
- Added benefit - accounts for contention caused by other processes of the *same* program

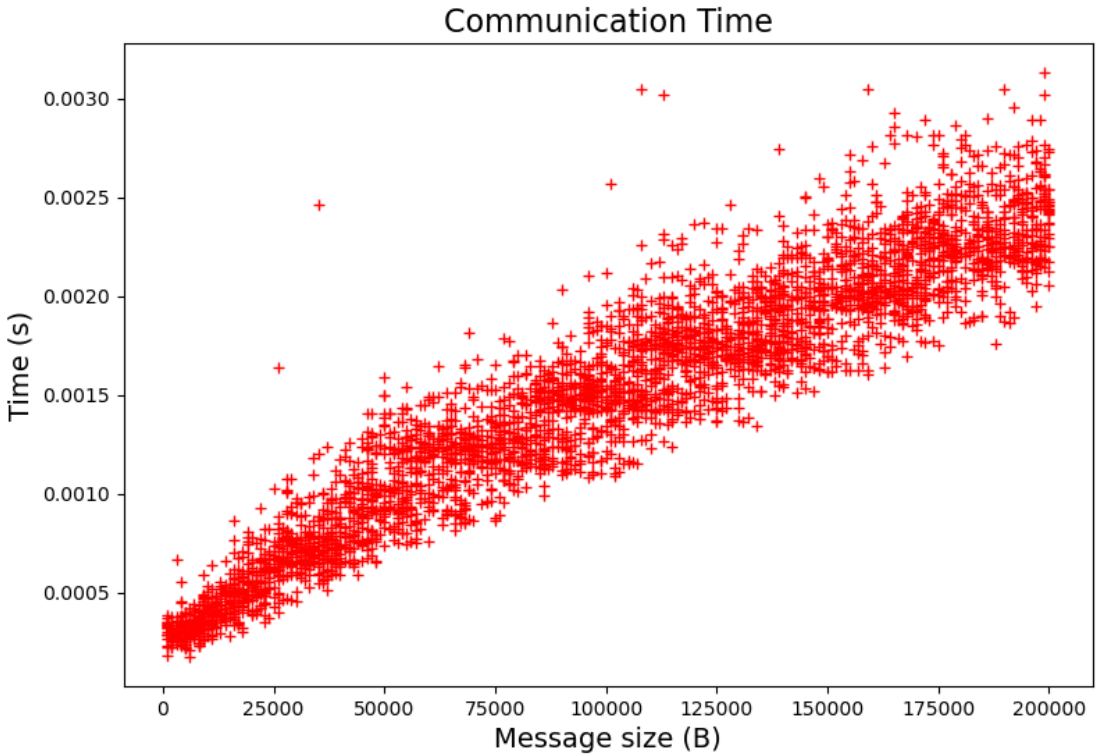
Strategy 2 - Accounting for contention

- Key idea - tokenize communication parameters (α, β)
- Measure parameters for each node rather than each pair
- Sends/receives messages at each node and generates CSVs
- Added benefit - accounts for contention caused by other processes of the *same* program
- Linearly interpolate above values to estimate time spent in a message

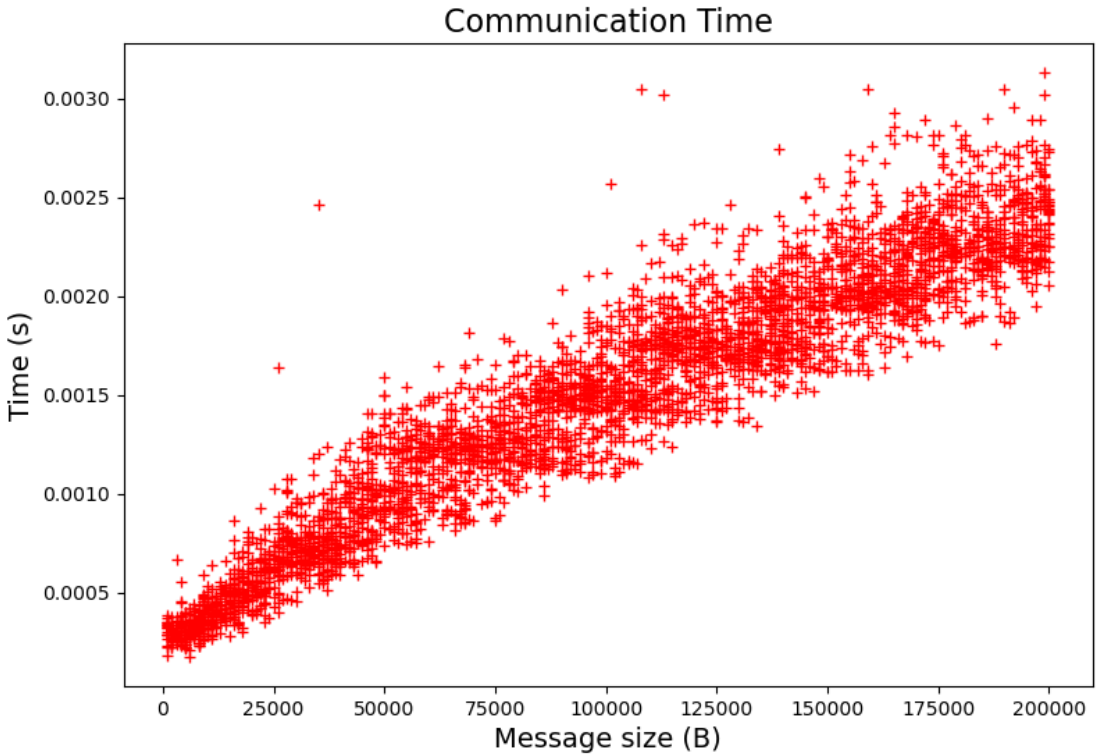
Strategy 2 - Accounting for contention

- Key idea - tokenize communication parameters (α, β)
- Measure parameters for each node rather than each pair
- Sends/receives messages at each node and generates CSVs
- Added benefit - accounts for contention caused by other processes of the *same* program
- Linearly interpolate above values to estimate time spent in a message
- Artificial congestion shows up

Model for P2P messages

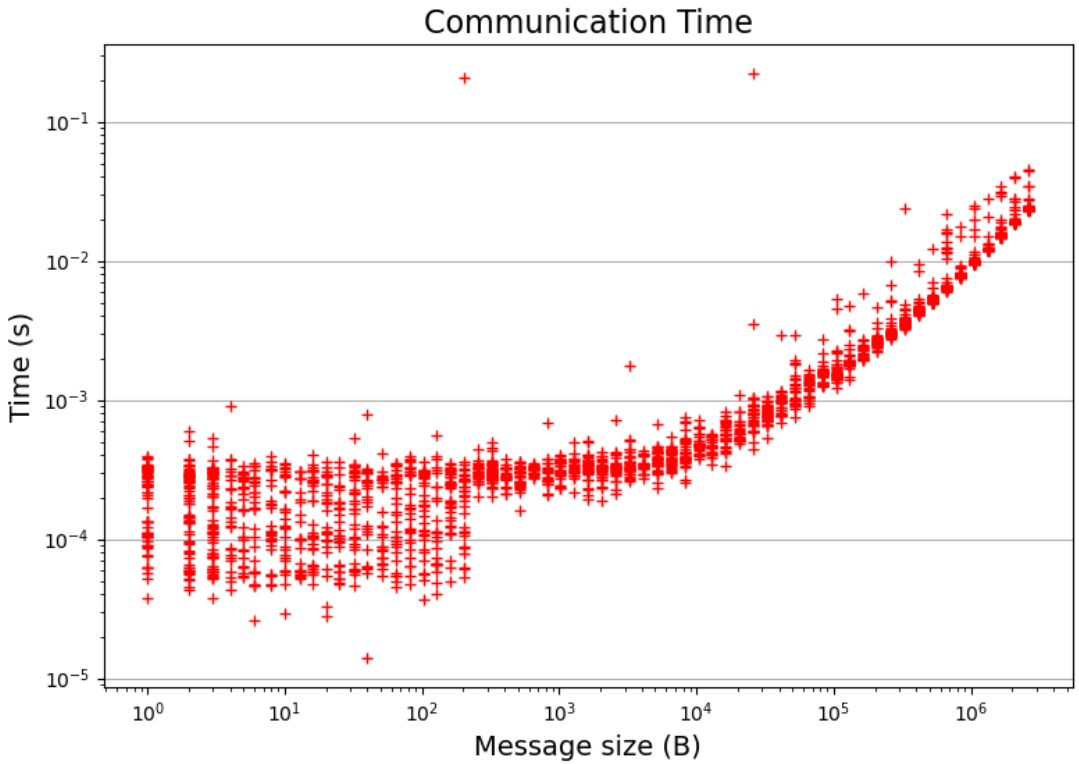


Model for P2P messages

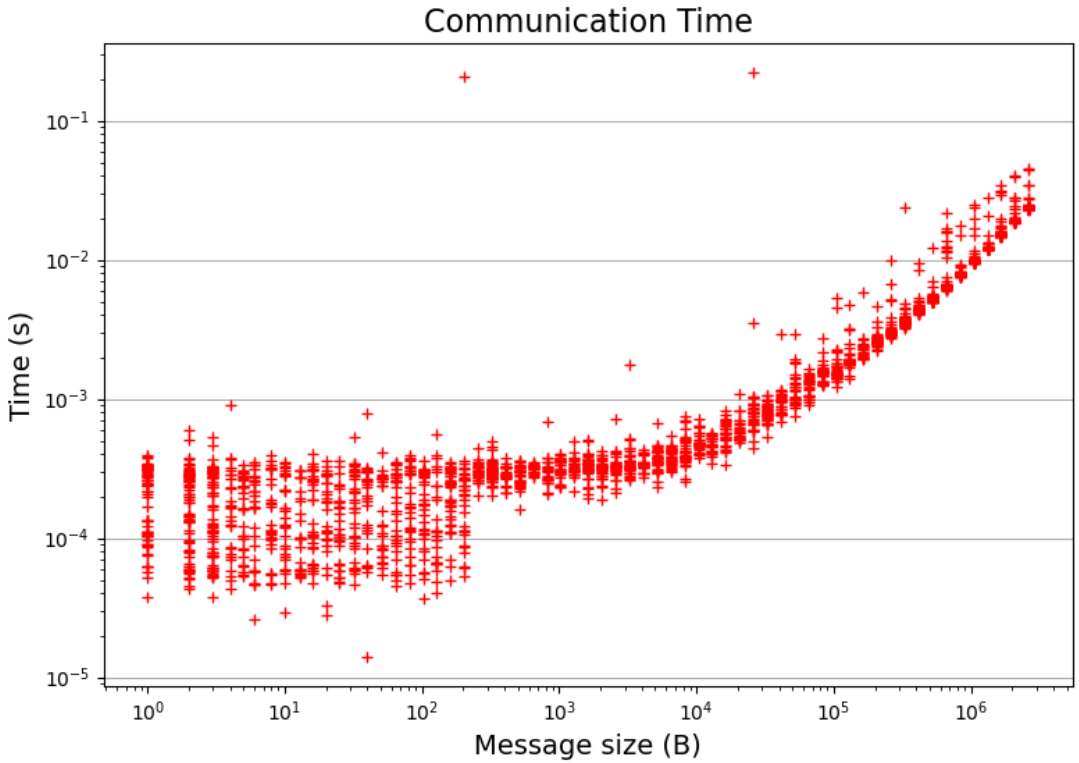


- Two linear parts, threshold 80-100 KB
- Separate Hockney parameters for messages smaller/larger than 128 KB

Model for P2P messages



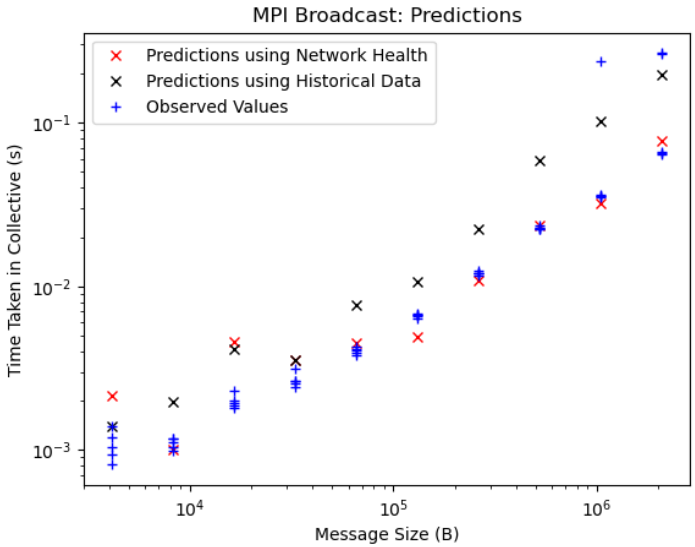
Model for P2P messages

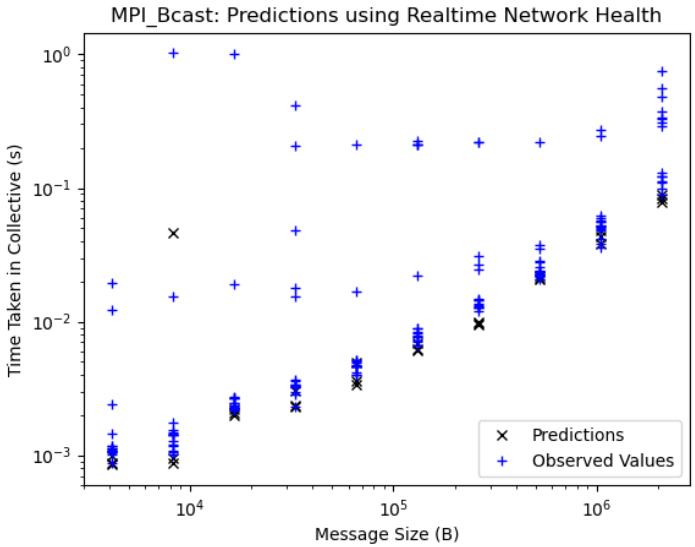


- Small messages take nearly constant time
- Assumed constant time for messages smaller than 8 KB
- Intranode messages - No pattern, used historical data

Results

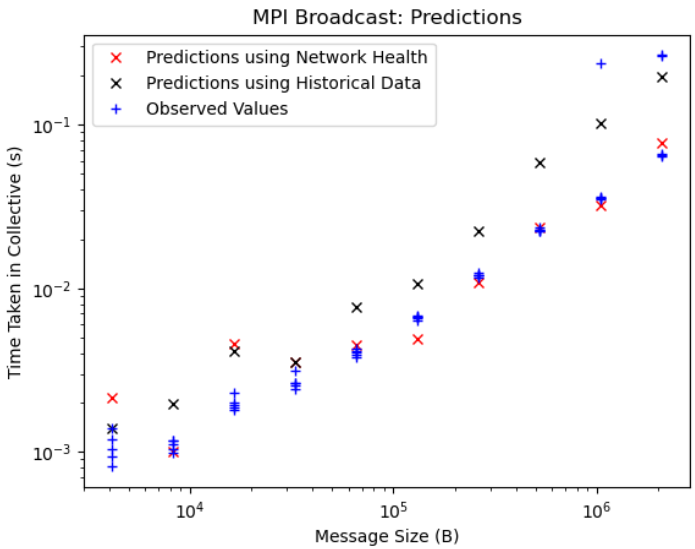
MPI Broadcast on 8 processes:

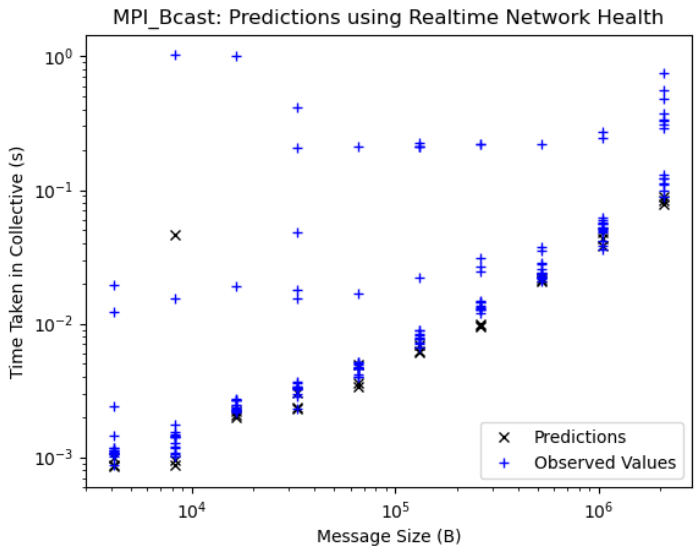




Results

MPI Broadcast on 8 processes:

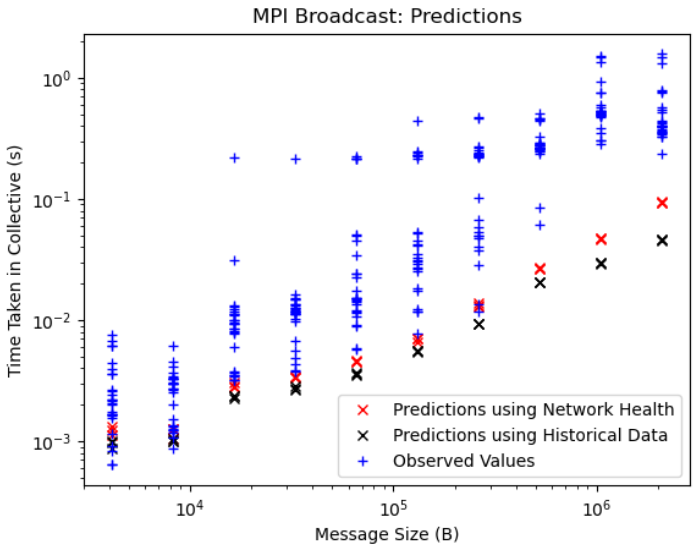


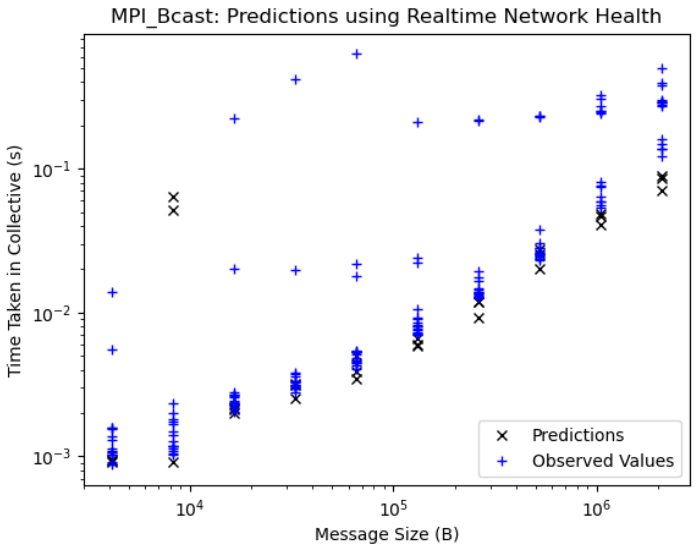


- Programs run on different days
- Noise affects quality of predictions

Results

Broadcast with artificial contention on 8 processes:





- Predictions using realtime data are usually higher

Results

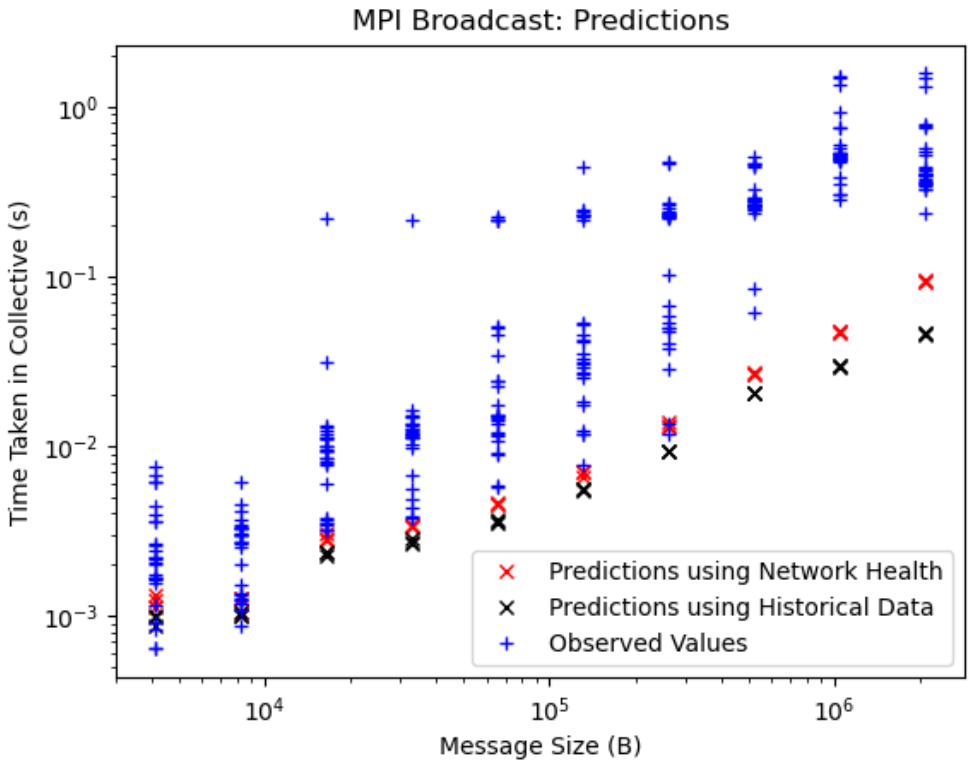


Figure: Broadcast on 16 processes (2 Processes Per Node) with artificial congestion

Results

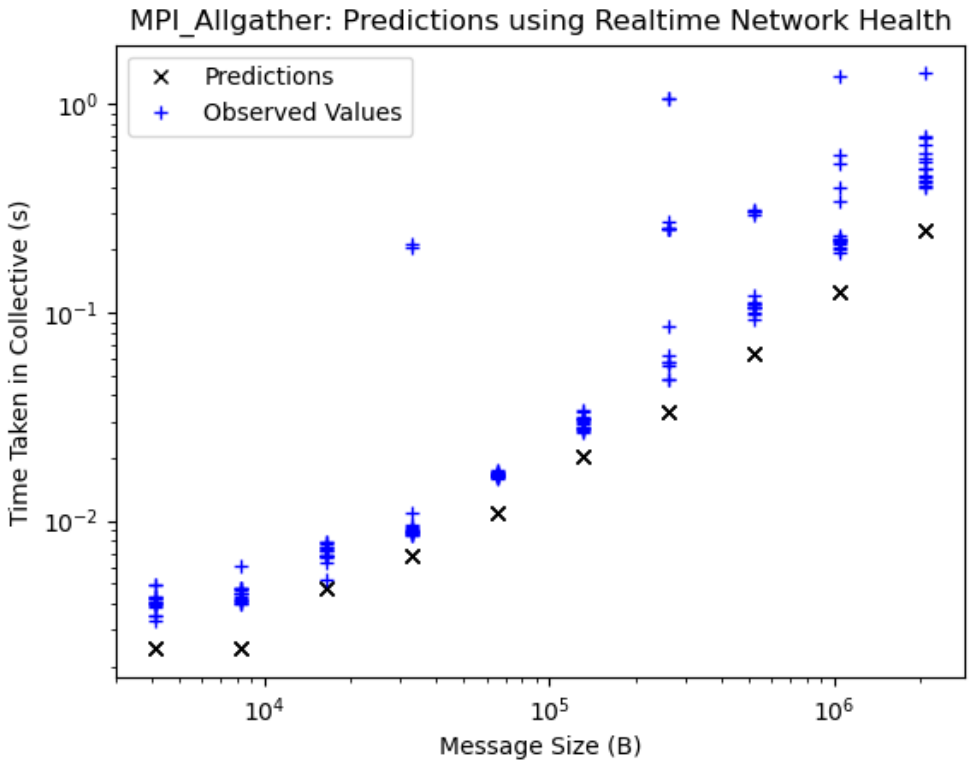


Figure: MPI- Allgather on 8 processes (1 PPN)

Results

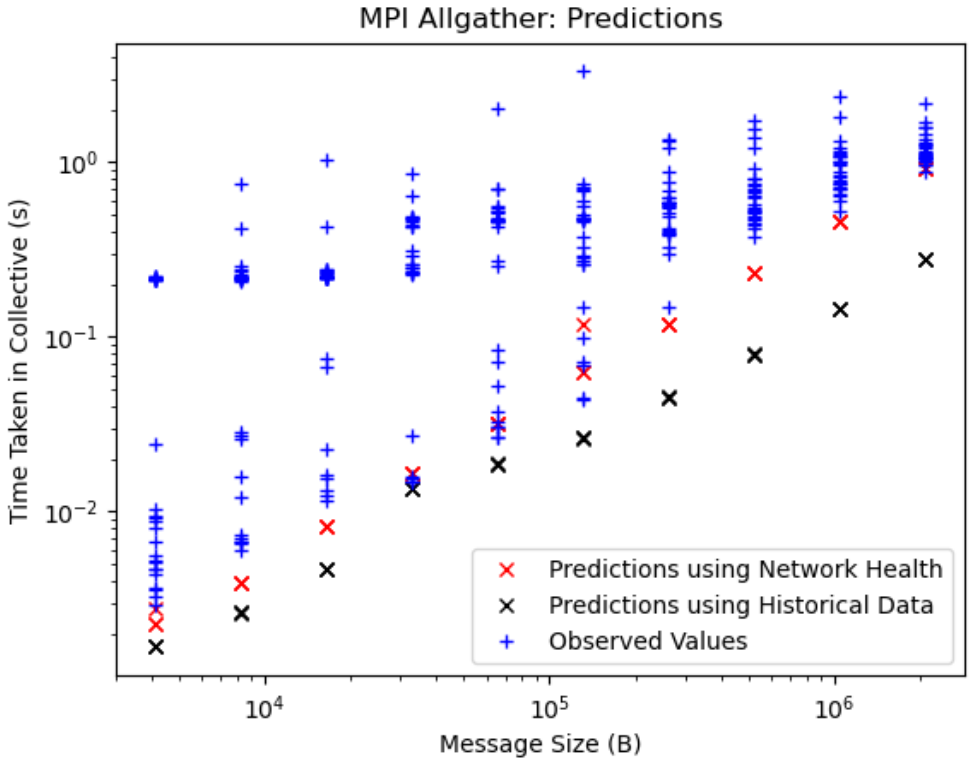


Figure: Allgather on 16 processes (2 PPN) with artificial congestion

Results

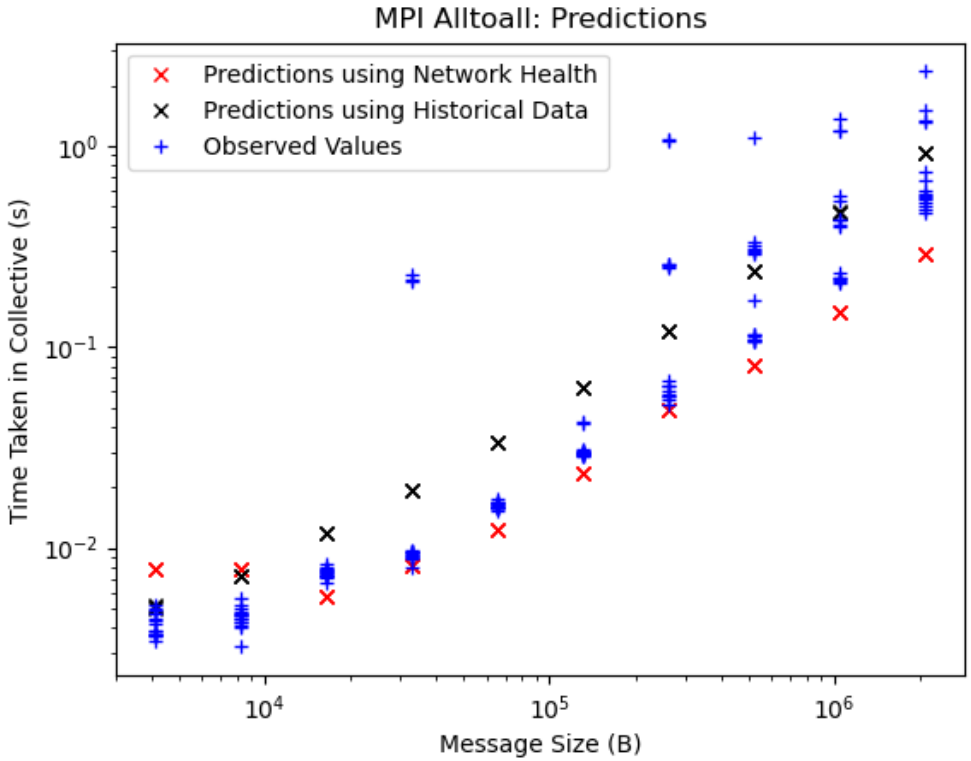


Figure: Alltoall on 8 processes (1 PPN)

Results - Alltoall

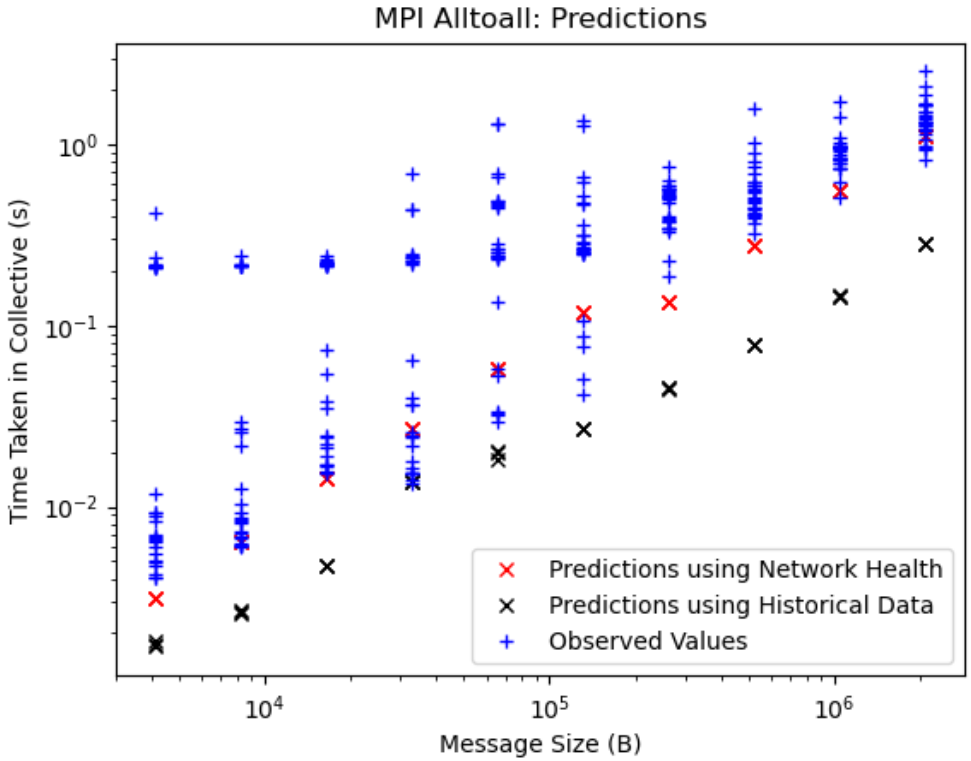


Figure: Allgather on 16 processes (2 PPN)

Tokenization Overhead

- Realtime communication parameters are found with 140 explicitly synchronised messages
- Takes under 1.5 s without contention, < 4 s with artificial congestion (8 nodes at a time, but can be increased)

IPMPI Overheads

- Our extensions to IPMPI read files and generate lookup tables
- File I/O requires significant time, but once for each run
- Typically < 1.5 s (both predictions) for one collective

Thank you

Questions?