Accurately Measuring Collective Operations at Massive Scale

Torsten Hoefler, Timo Schneider, Andrew Lumsdaine

Open Systems Lab
Indiana University
Bloomington, USA

IPDPS'08 - PMEO'08 Workshop
Miami, FL, USA
April, 18th 2008

Introduction

- network performance measurement and prediction is important
- assess the runtime of parallel algorithms
- optimize communication patterns (e.g., collective)
- important for application programmers to choose collectives

The approach

Accurately measure collective communication to derive and test abstract models. Specialized models for hardware-supported collectives.

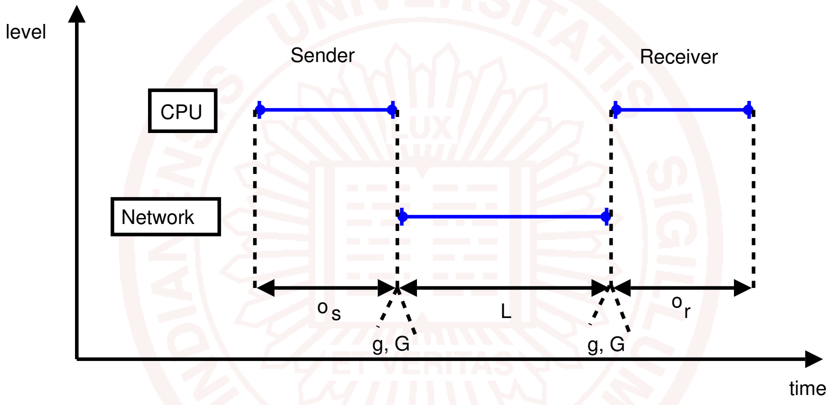
Introduction

- network performance measurement and prediction is important
- assess the runtime of parallel algorithms
- optimize communication patterns (e.g., collective)
- important for application programmers to choose collectives

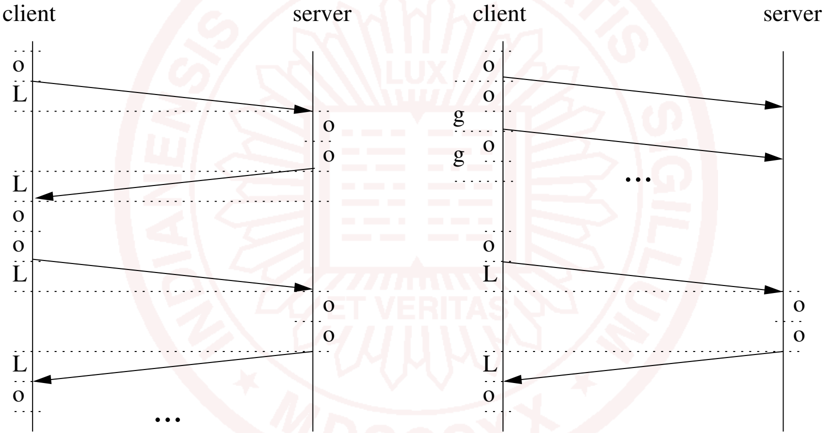
The approach

Accurately measure collective communication to derive and test abstract models. Specialized models for hardware-supported collectives.

The LogGP Model



no central clock → measurements on one host only



First steps ... towards accurate modelling

- measure communication performance correctly
- some general hints given by Gropp et al. in “Reproducible Measurements of MPI Performance Characteristics” and Worsch et al. “On Benchmarking Collective MPI Operations”

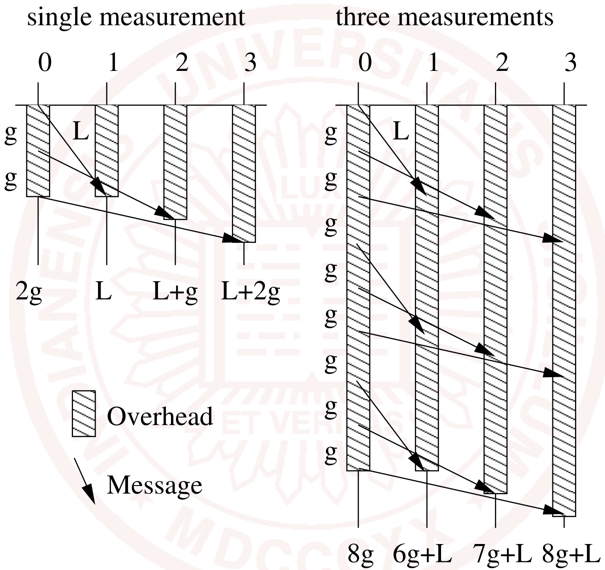
Common Mistakes

- merging results on multiple processes incorrectly
- pipelining effects in measurements
- process skew during measurements
- synchronization perturbation and congestion
- network congestion

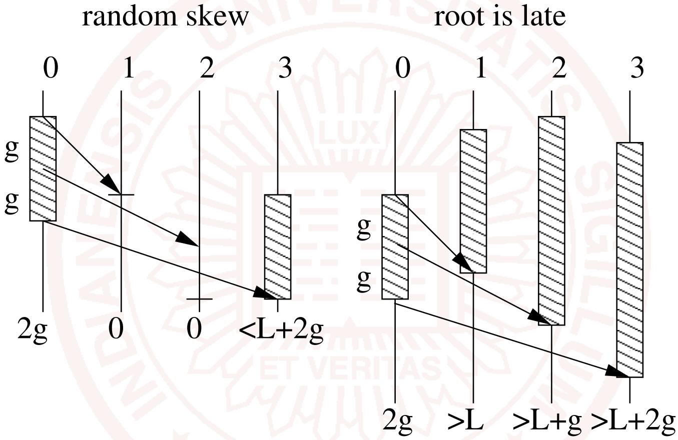
look at the following code-fragment from the MPP benchmark:

```
MPI_Gather(...); /* warmup */
MPI_Barrier(...); /* synchronization */
t0 = MPI_Wtime(); /* take time */
for (i=0; i<reps; i++) {
  MPI_Gather(...); /* execute benchmark */
}
t1 = MPI_Wtime(); /* take time */
MPI_Barrier(...);
time = t1-t0;
if (rank == 0) report_time();
```

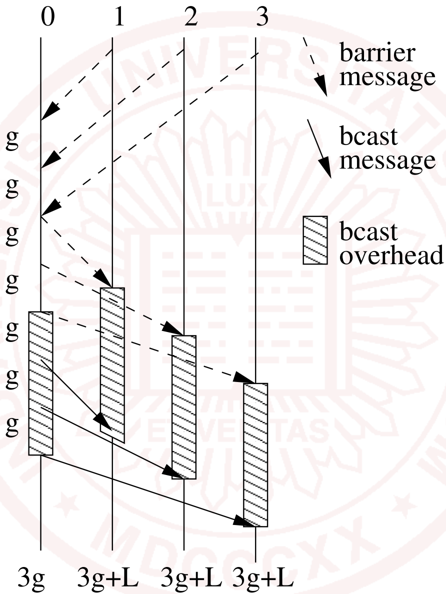
“pipelining effects in measurements”



“process skew during measurements”



“synchronization perturbation and congestion”



That are all the problems and

what

Is the solution?

A scheme similar to SKaMPI!

⇒ with some changes though ...

How does it work?

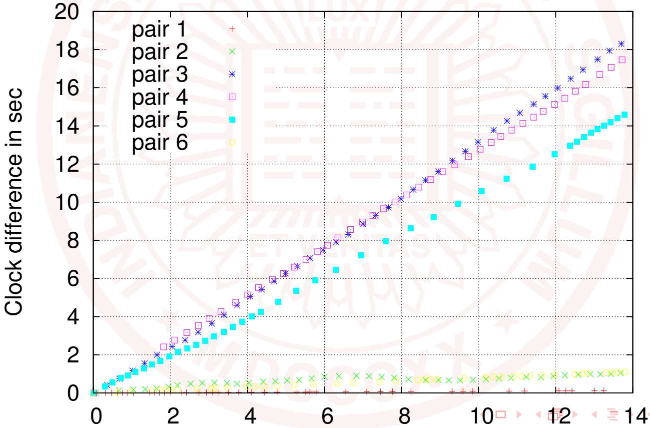
- processes “synchronize” to get a global time
- a designated process broadcasts a start time in the future and a window size
- all processes start at the same global time and run *n* benchmarks in *n* time-windows
- benchmarks that took longer than the window or started late are discarded
- window-size is determined adaptively at runtime

But ... I think it does not work!?

- time-skew in cluster systems might be too big/instable
- determination of the global time is inaccurate
- much time could be wasted for synchronization/in windows

"time-skew in cluster systems might be too big/instable"

- we "benchmarked" CPU clock counters (Netgauge)
- 50,000 measurements, once every second → 14 hrs
- ⇒ yes, the clocks drift! But linearly (correctable :)



“determination of the global time is inaccurate”

- SKaMPI employs a linear scheme where RTT/2 is substracted from the remote time
- this might be problematic
  1. is not scalable to high CPU counts
  2. the network latency varies (jitter)
- $\Rightarrow$ we propose a tree-based scheme with accurate point-to-point time synchronization

Point-to-point synchronization

Requirements

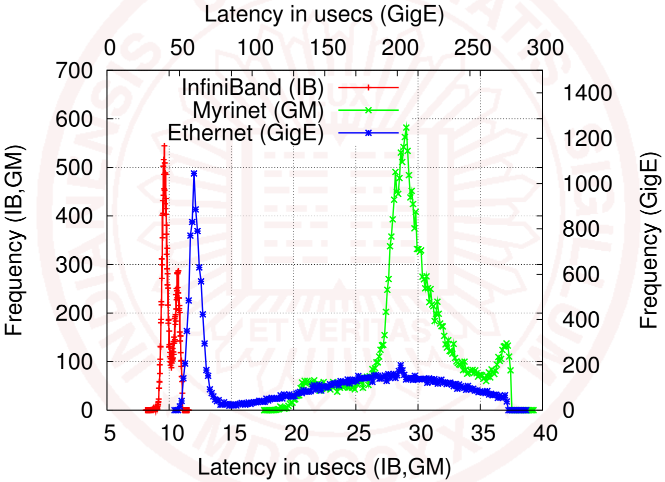
- must accept high network jitter
- must be fast (measurements depend adaptively on jitter)

Solution

- measure round-trip times *and* clock differences at the same time
- use only measurements that are below a certain (latency) threshhold
- threshold has to be determined dynamically (adaptive to jitter)
- $\Rightarrow$ repeat measurement until N successive measurements dont have a smaller latency than the smallest

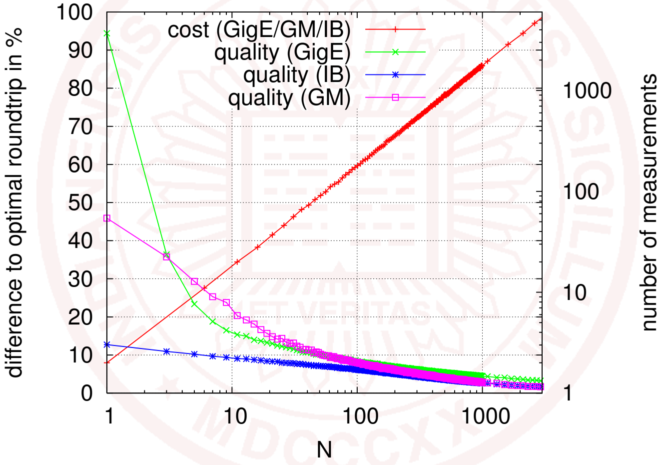
How does that work?

- let's take a look at message latency distributions
- recycle our 50,000 measurements from before



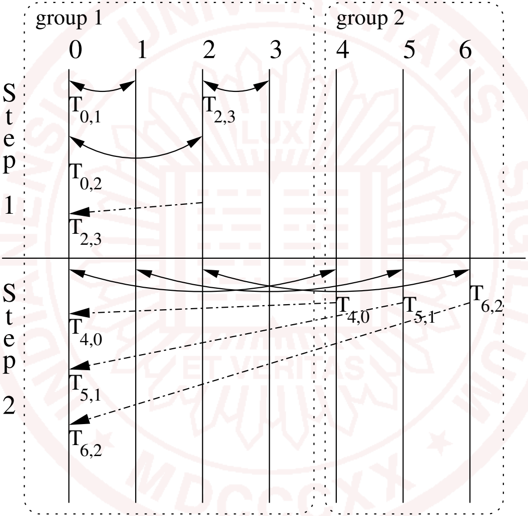
What should we choose as *N*?

- choice of *N* is non-trivial
- modeling not easily possible :-(→ simulation

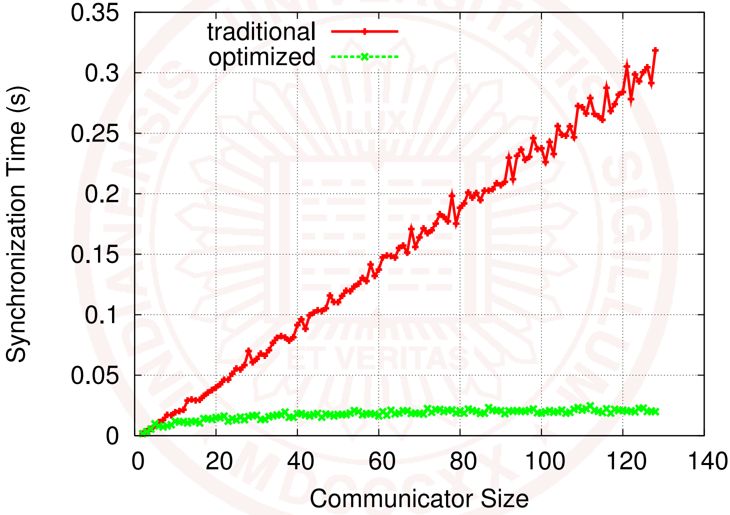


“much time could be wasted for synchronization”

- we propose tree-based scalable synchronization!



- errors propagate logarithmically :-(... but it's fast:



Conclusions and Future Work

Conclusions

- improved collective benchmarking (NBCBench)
- improved time-synchronization scheme
- adaptive point-to-point synchronization

Future Work

- verify collective operation models (Grbovic et. al.)
- models for non-blocking collective operations

Conclusions and Future Work

Conclusions

- improved collective benchmarking (NBCBench)
- improved time-synchronization scheme
- adaptive point-to-point synchronization

Future Work

- verify collective operation models (Grbovic et. al.)
- models for non-blocking collective operations