Automatic Construction of Coordinated Performance Skeletons

(Predicting Performance in an Unpredictable World)

Jaspal Subhlok **Qiang Xu**

University of Houston

NSFNGS 2008

Getting Started

OBJECTIVE: Estimate application performance rapidly in a foreign/dynamic environment, e.g

- Cluster with upgraded hardware or software components, e.g., MPI Library
- Desktop grid or "Volunteer nodes" or Amazon EC-2 with a shared network
- Execution with different number of processes (8,16 or more processes best for 8 nodes)
- System under simulation

Motivated by resource selection, mngmt, etc.

Skelton Based Approach ?

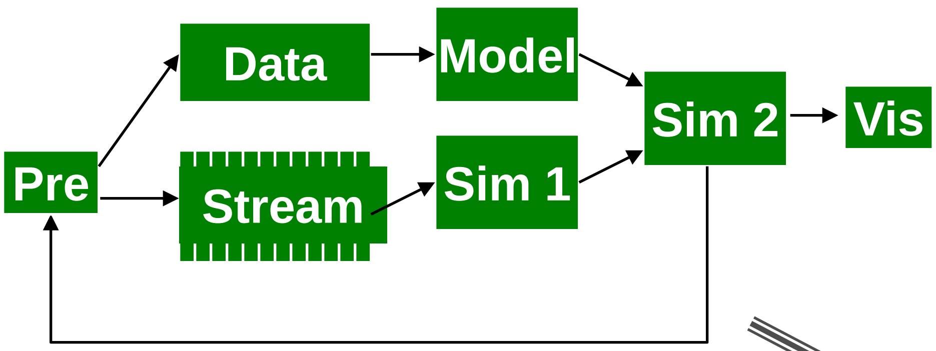
Build a short running "*skeleton*" program that mimics execution behavior of a given application

GOAL: execution time of a performance skeleton is a fixed fraction of application execution time - say 1:1000, then..

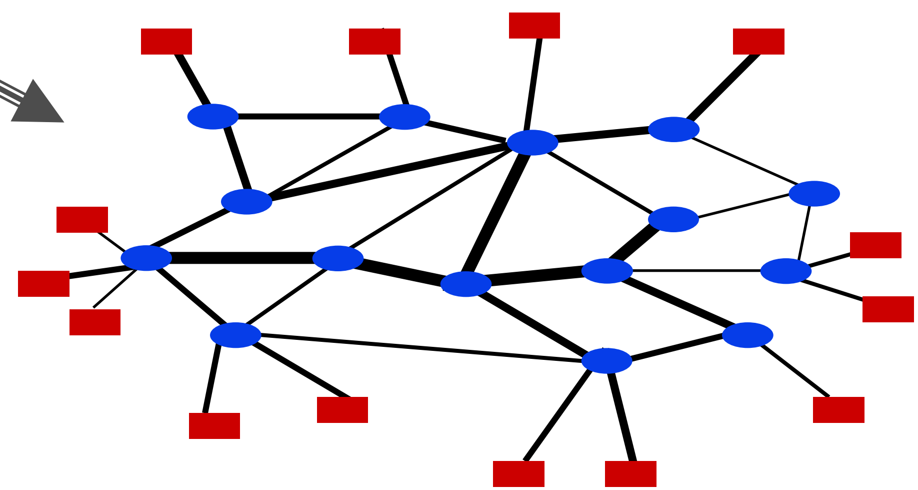
If the Application runtime is	**Skeleton runs in**
10K seconds on a dedicated compute cluster	10 secs
8K seconds with Open MPI on that cluster	8 secs
20K seconds on a shared heterogeneous grid	20 secs
1 million seconds under simulation	1000 secs
1K seconds on a supercomputer	1 second
…..,	

Timed execution of performance skeleton provides an estimate of application performance!

One Motivation: Mapping Distributed Applications on Networks



Application

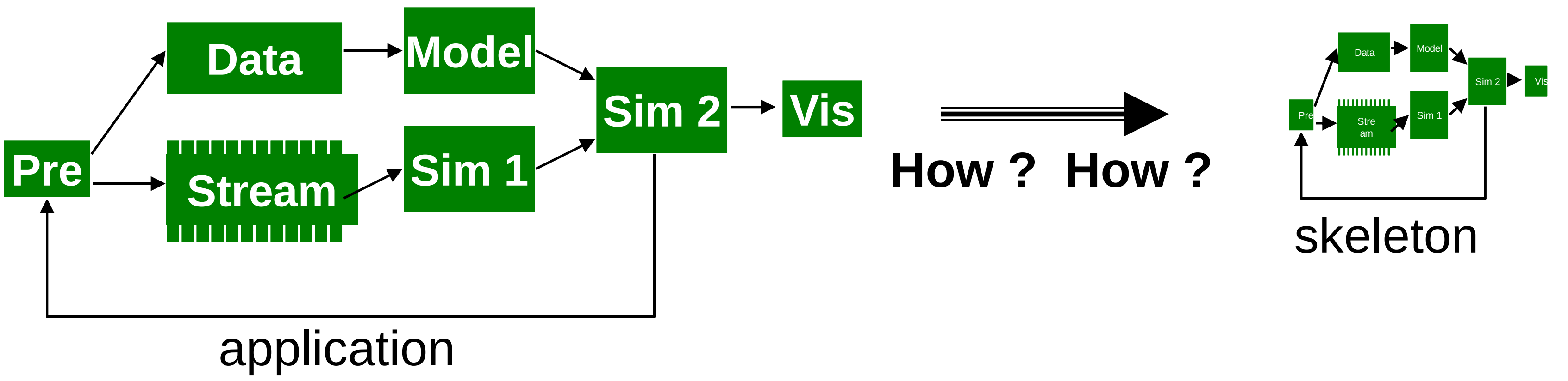


Network

Predict performance and select nodes by actual execution of performance skeletons on groups of nodes **?**

CS@UH

How to Construct a Performance Skeleton ?



Central challenge in this research

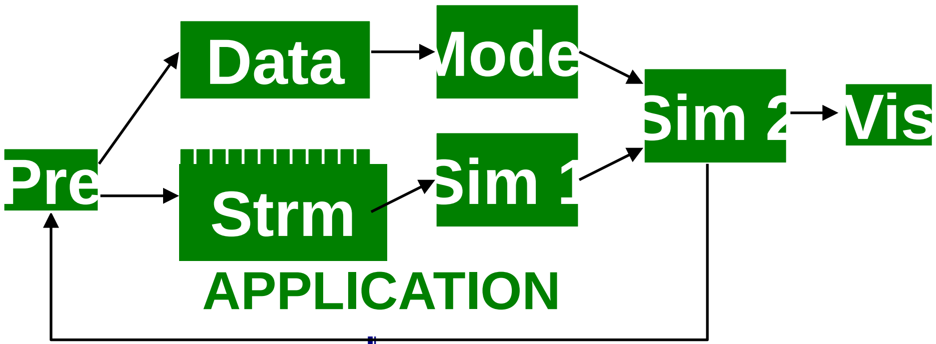
Common sense dictates that an application and its skeleton must be similar in:

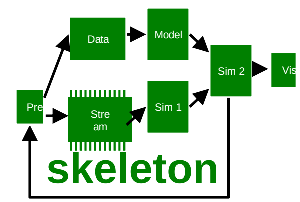
- **Computation behavior**
- **Communication behavior**
- **Memory behavior**
- I/O Behavior

All execution behavior is to be captured in a **short program**

Skeleton Construction

Implementation for parallel MPI codes





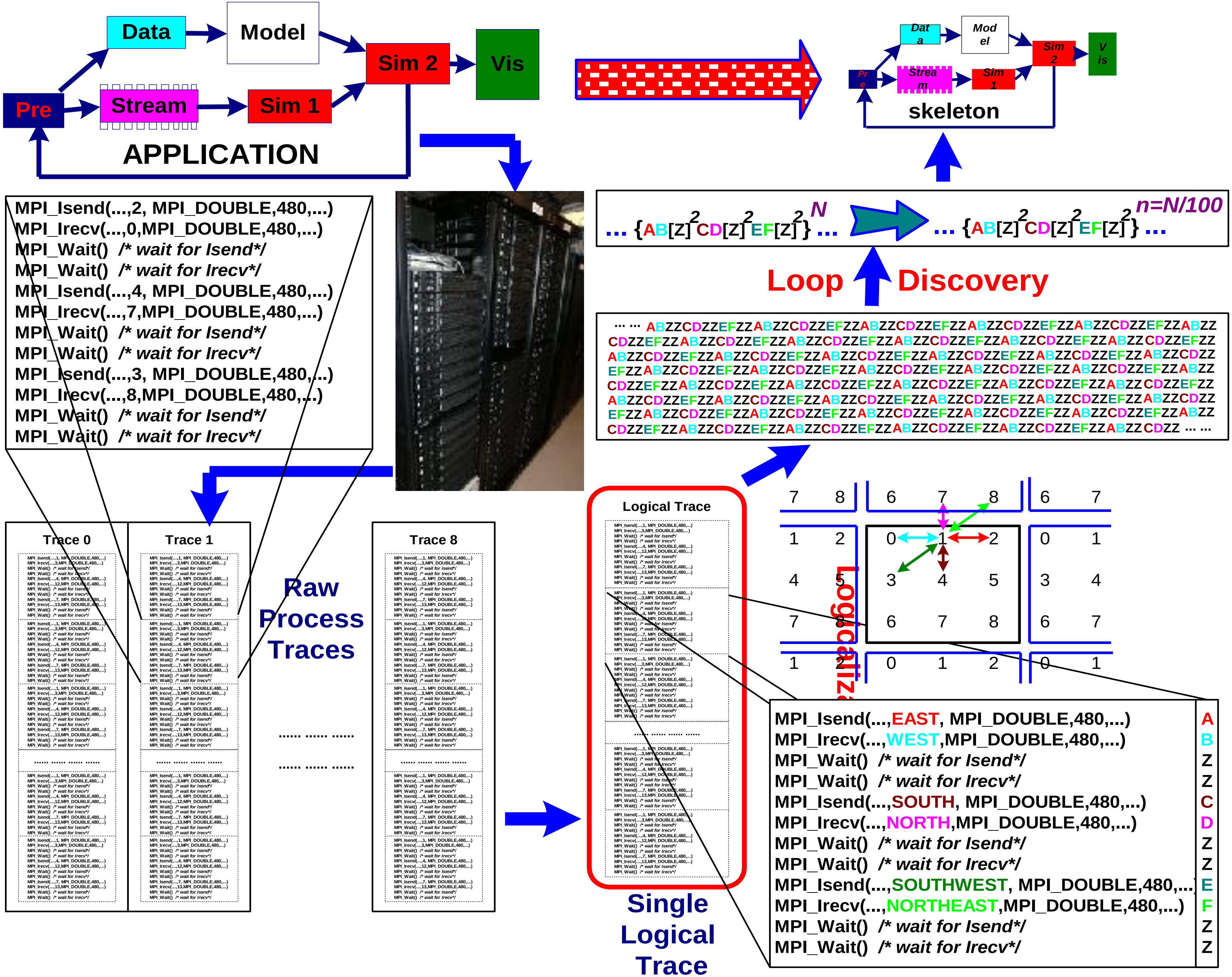
Record Physical Execution Traces

Logicalize Physical Traces into a Single Logical Trace

Compress the Logical Trace into Compact Execution Signature

Construct Executable Performance Skeleton

CS@UH



Data
Model
Sim 2
Vis
Pre
Stream
Sim 1
APPLICATION
Pre
Data
Model
Sim 2
Vis
Stream
Sim 1
skeleton
MPI_Isend(...,2, MPI_DOUBLE,480,...)
MPI_Irecv(...,0,MPI_DOUBLE,480,...)
MPI_Wait() /* wait for Isend*/
MPI_Wait() /* wait for Irecv*/
MPI_Isend(...,4, MPI_DOUBLE,480,...)
MPI_Irecv(...,7,MPI_DOUBLE,480,...)
MPI_Wait() /* wait for Isend*/
MPI_Wait() /* wait for Irecv*/
MPI_Isend(...,3, MPI_DOUBLE,480,...)
MPI_Irecv(...,8,MPI_DOUBLE,480,...)
MPI_Wait() /* wait for Isend*/
MPI_Wait() /* wait for Irecv*/
... {AB[Z]^2CD[Z]^2EF[Z]^2}^N ...
... {AB[Z]^2CD[Z]^2EF[Z]^2} ...
n=N/100
Loop Discovery
... ... ABZZCDZZEFZZABZZCDZZEFZZABZZCDZZEFZZ ... ...
Trace 0
Trace 1
Trace 8
Raw Process Traces
Logical Trace
Logicalization
Single Logical Trace
MPI_Isend(...,EAST, MPI_DOUBLE,480,...) A
MPI_Irecv(...,WEST,MPI_DOUBLE,480,...) B
MPI_Wait() /* wait for Isend*/ Z
MPI_Wait() /* wait for Irecv*/ Z
MPI_Isend(...,SOUTH, MPI_DOUBLE,480,...) C
MPI_Irecv(...,NORTH,MPI_DOUBLE,480,...) D
MPI_Wait() /* wait for Isend*/ Z
MPI_Wait() /* wait for Irecv*/ Z
MPI_Isend(...,SOUTHWEST, MPI_DOUBLE,480,...) E
MPI_Irecv(...,NORTHEAST,MPI_DOUBLE,480,...) F
MPI_Wait() /* wait for Isend*/ Z
MPI_Wait() /* wait for Irecv*/ Z

CS@UH

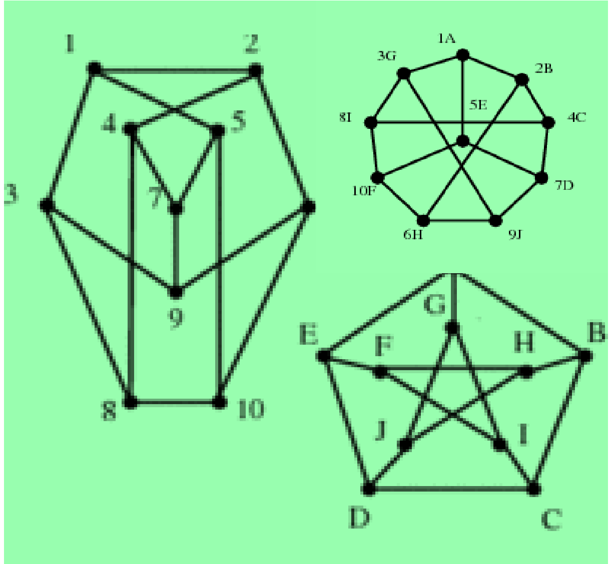
Logicalization

Key challenge: Identify the dominant communication topology from pairwise node communication matrix

Matching against a known topology
Is solving graph isomorphism
- *No polynomial algorithm*

Practical solution employed:

1. **Match node & edge counts**
2. **Match eigenvalues**
3. **Graph Isomorphism algorithm**



First two test eliminate most patterns but cannot prove a match. Exact test used sparingly.

Logicalization Notes

Works well in practice!

- Main communication topology must be static and regular
- Matching only against known patterns, but patterns easy to add and library can be large
  - All n-dim grids or n-ary trees specified in one shot
- Some message exchange not related to main communication pattern observed
  - Ignored with thresholding
  - Can cause innacuracy, reported to user
- Multiple mixed patterns (equal to subgraph isomorphism) not yet implemented

Compression of Logical Trace

Goal is to identify loop nests in the trace!

Matching sliding windows of trace is $O(N^3)$.
Commonly employed locally on trace sections
So can miss long range repeats (outer loops).

Two new algorithms developed:

1. An optimal $O(N^2)$ algorithm (finds outer loops first) : leverages Crochemore's algorithm to find all repeats
2. Greedy algorithm (finds inner loops first) guaranteed to miss at most 2 iterations of a loop – Very fast

Loop Discovery Performance

	Raw Trace Length (MPI Calls)	Compressed Trace Length (MPI Calls)	Optimal Loop Discovery (seconds)	Greedy Loop Discovery (seconds)
BT	17106	44	311.18	8.91
SP	26888	89	747.73	7.61
LU	323048	63	*113890.21 (~30 hours)*	*61.9*
CG	41954	10	240.27	8.48
MG	10047	648	144.54	10.88

CS@UH

Validation of Skeleton Construction

Skeletons constructed for Class C NAS MPI benchmarks up to 128 nodes

Skeletons employed to predict performance in a variety of new scenarios

- **Execution with different number of nodes for the same number of processes**
- **Execution under varying available bandwidth**
- **Execution under competition with other jobs**
- **Execution on a different clusters**
- **Execution under a new MPI library (Open MPI)**

Validation Results

Skipping the large suite of graphs!

For most applications and scenarios, the prediction was rather accurate with error within 10% for skeletons running for a few minutes

However:

- Prediction with competing jobs inaccurate!
- Some scenarios showed high errors (> 20%) in particular CG benchmark.

Reasons:

1. Computing not modeled precisely (memory, instructions)
2. Synchronization impact can exaggerate variations

Conclusions

p

- Performance skeletons are an effective tool for estimating performance dynamically

- Methodologies for logicalization and loop nest discovery have broad applicability

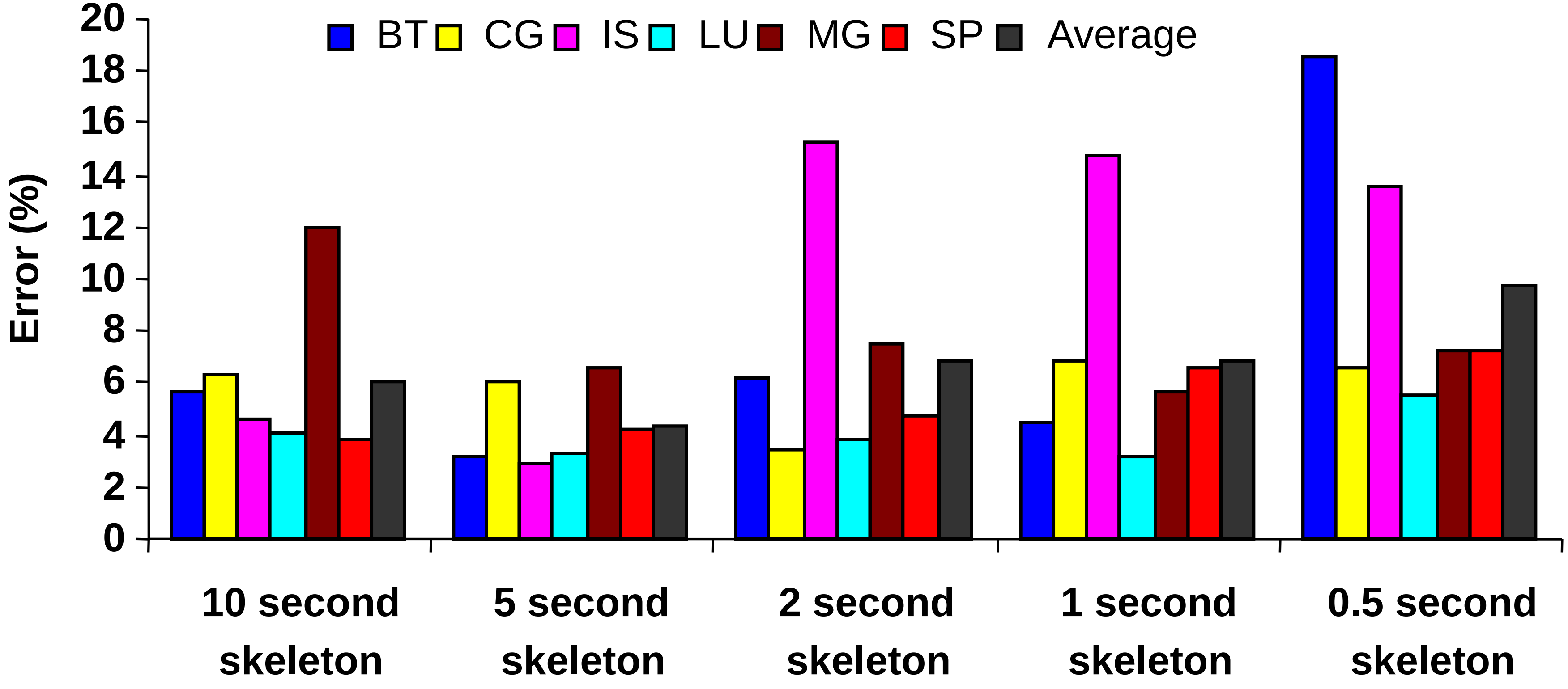
Open to collaborations!
Thanks to NSF

FOR MORE INFORMATION:

- **www.cs.uh.edu/~jaspal** **jaspal@uh.edu**

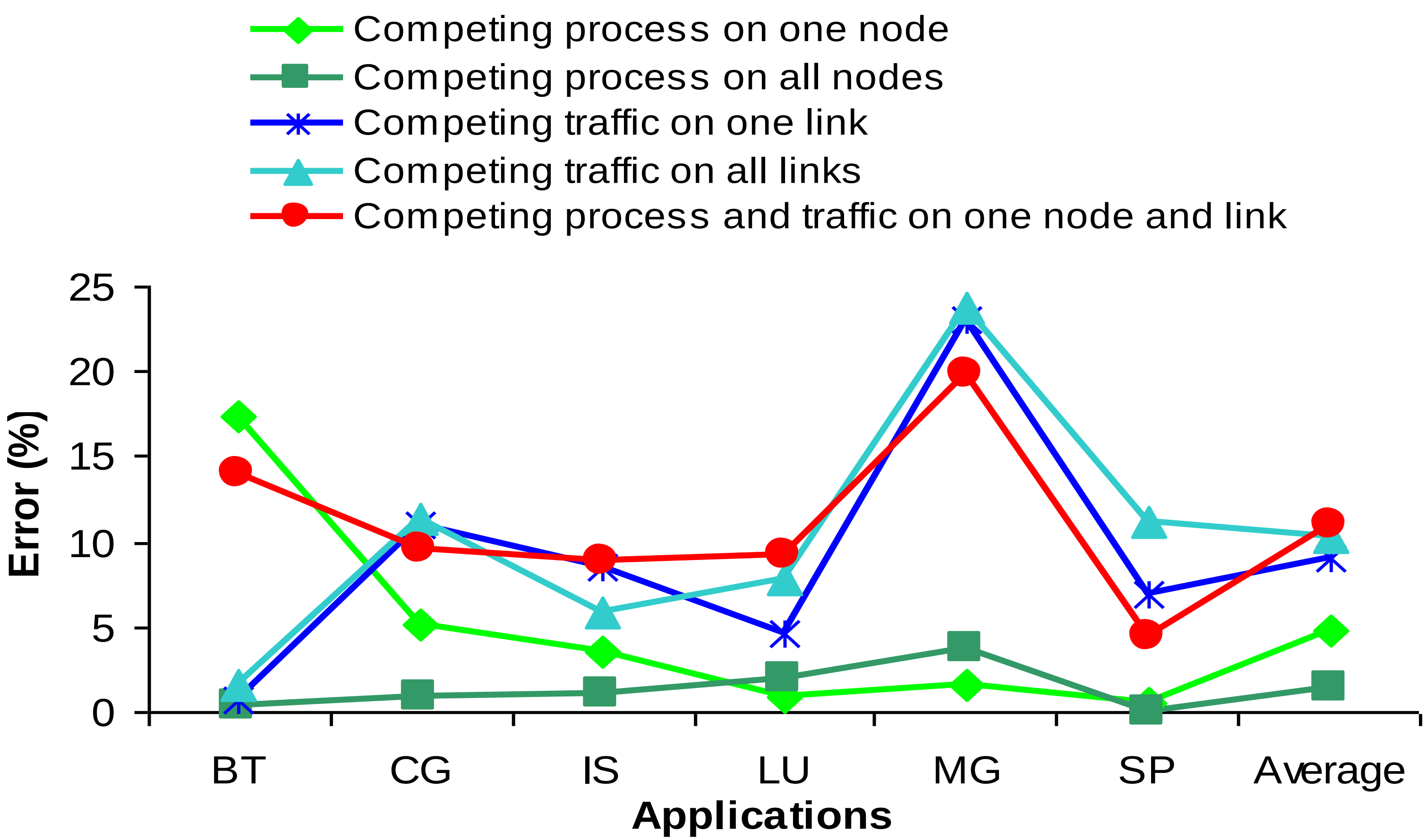
Prediction Accuracy of Skeletons

(average across all sharing scenarios)



CS@UH

Prediction for Different Sharing Scenarios
(10 second skeletons)

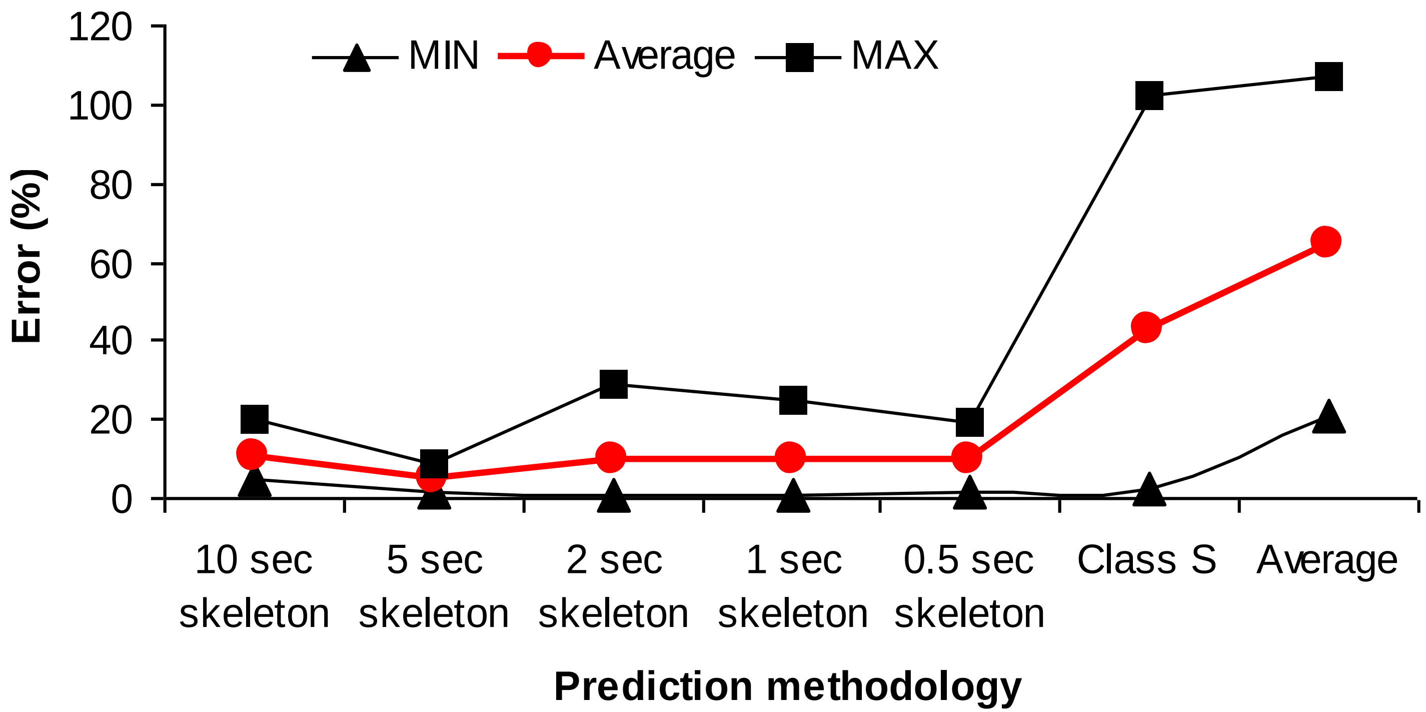


Error is higher with network contention

- **communication is harder to scale down and affects synchronization more directly**

CS@UH

Comparison with Simple Prediction Methods



Average Prediction: Average slowdown of entire benchmark used to predict execution time for each program.

Class S Prediction: Class S benchmark(~1sec) programs used as skeletons for Class B (30-900s)benchmarks

Even the smallest skeletons are far superior!