A Lightweight Java Taskspaces Framework for Scientific Computing on Computational Grids

H. De Sterck[1], R.S. Markel[2], T. Pohl[3], and U. Rüde[3]

[1]Department of Applied Mathematics, University of Colorado at Boulder, USA

[2]Advansys, Inc., Boulder, CO, USA

[3]Department of Computer Science, University of Erlangen-Nürnberg, Germany

SAC 2003 - PDSN

Tuesday, March 11, 2003

(desterck@colorado.edu)

Introduction

Taskspaces grid computing framework:

- Objectives:
  1. heterogeneous grids: use isolated workstations, Beowulf clusters, parallel supercomputers, connected over the internet, “all” operating systems
  2. lightweight, usability (user – administrator): copy one single small executable file to worker and execute one single command to start worker; all application code downloaded
  3. applications: taskfarming, but also “parallel scientific computing” with intertask communication
- Other approaches:

Condor – Globus – JavaSpaces – Entropia – TurboWorx

Outline

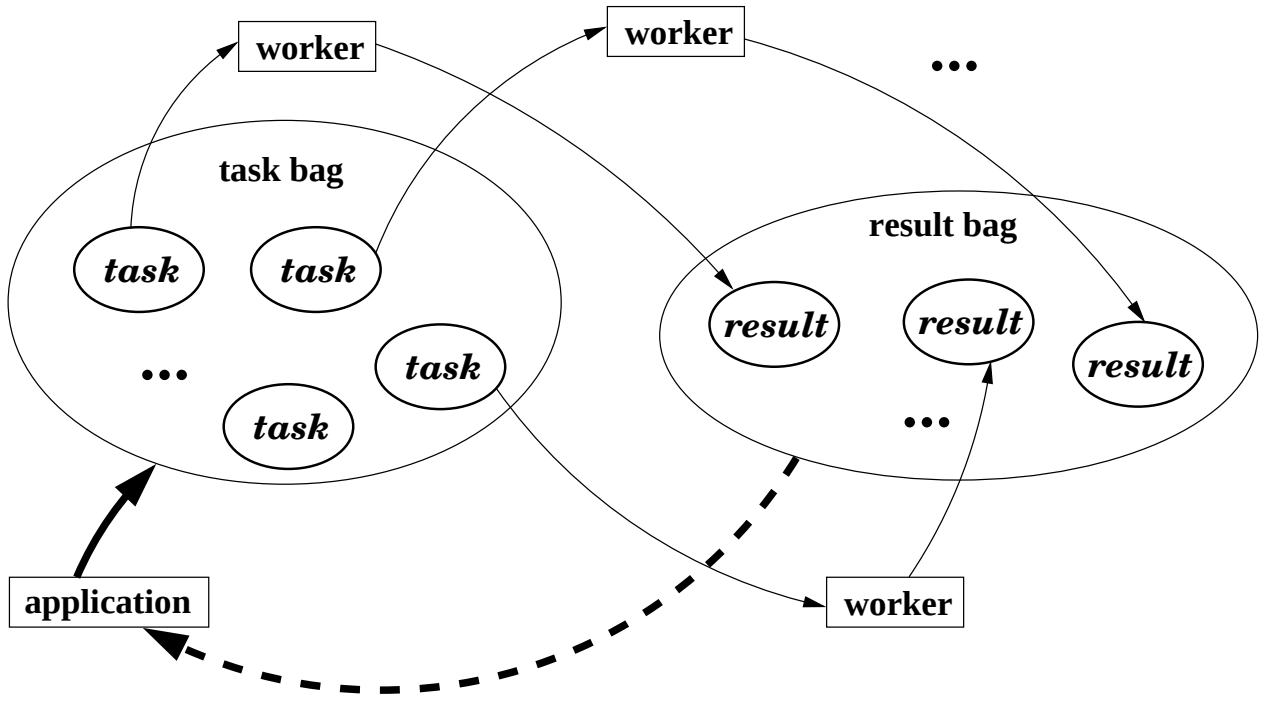
1. Design and implementation
2. Parallel scaling tests
   - Jacobi iteration
3. Other applications
4. Future work

1. Design and implementation

- design principles

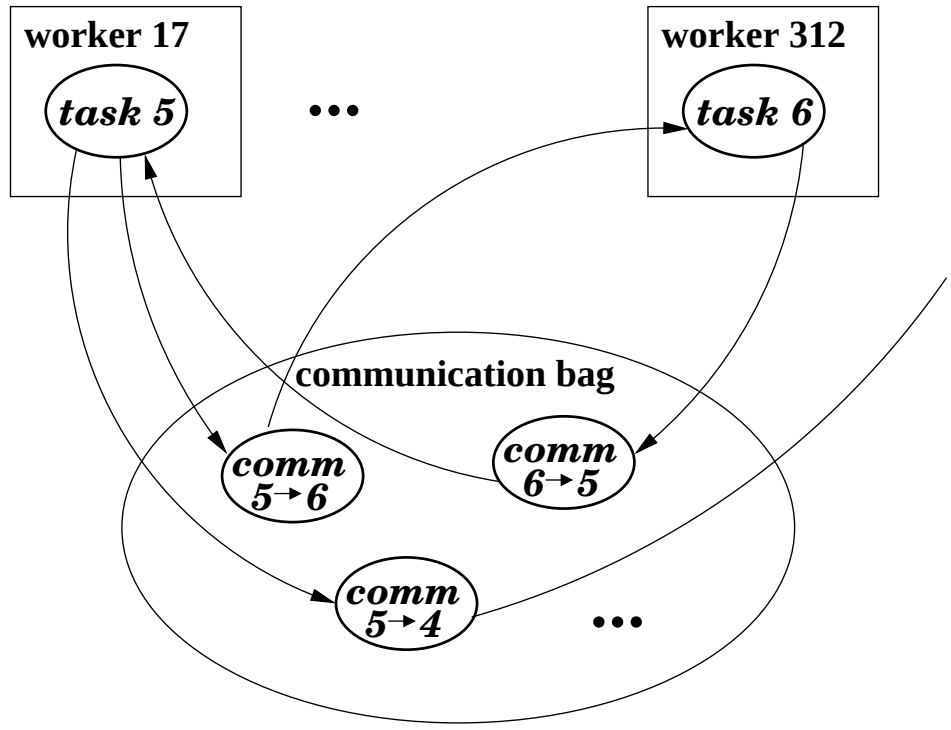
  1. decentralization through the use of tuple spaces
  2. distributed tuple space for intertask communication
  3. platform independence, object orientation, code downloading, security mechanisms $\Rightarrow$ Java
  4. mobile agent objects for grid administration

Tuple spaces and bag-of-tasks paradigm



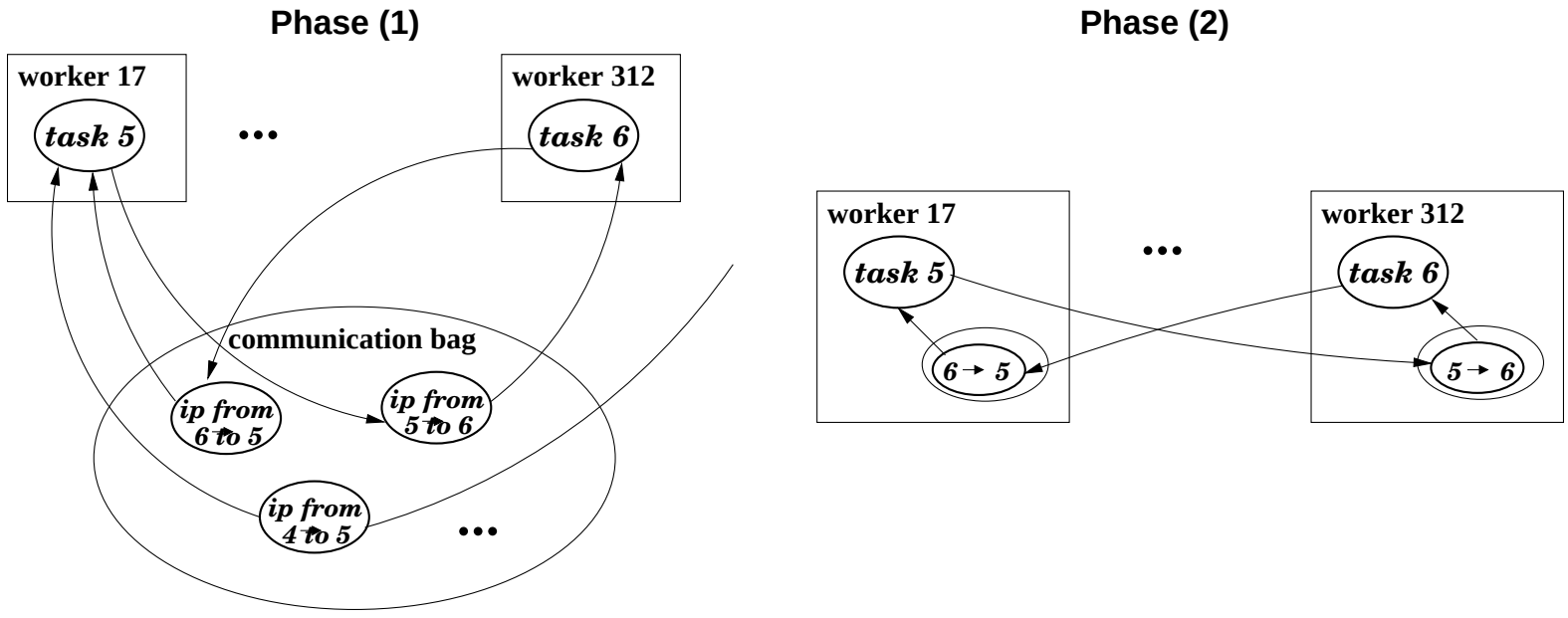
- a Tuple Space contains Task Objects
- Task Object = data + methods (code)
- decoupled in space and time
- self-configuring, no central coordination
- fault-tolerance (transaction–based)
- Tuple Spaces pioneered in the late 70s (Gelernter, “Linda”)

Communication – global tuple space



global communication space = bottleneck

Communication – distributed tuple space

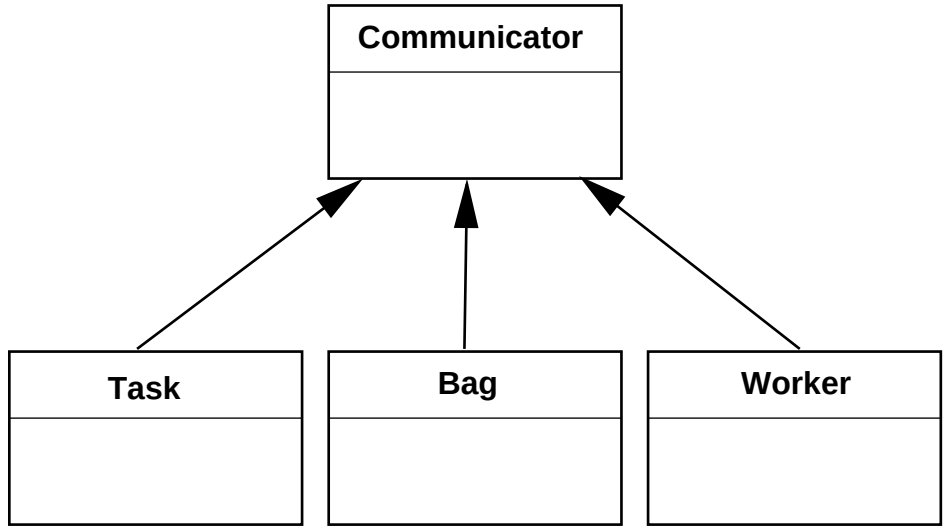


- distribute the communication tuple space over the workers
- two stages

  (1) set up communication pattern through global tuple space (only one time for fixed communication pattern)

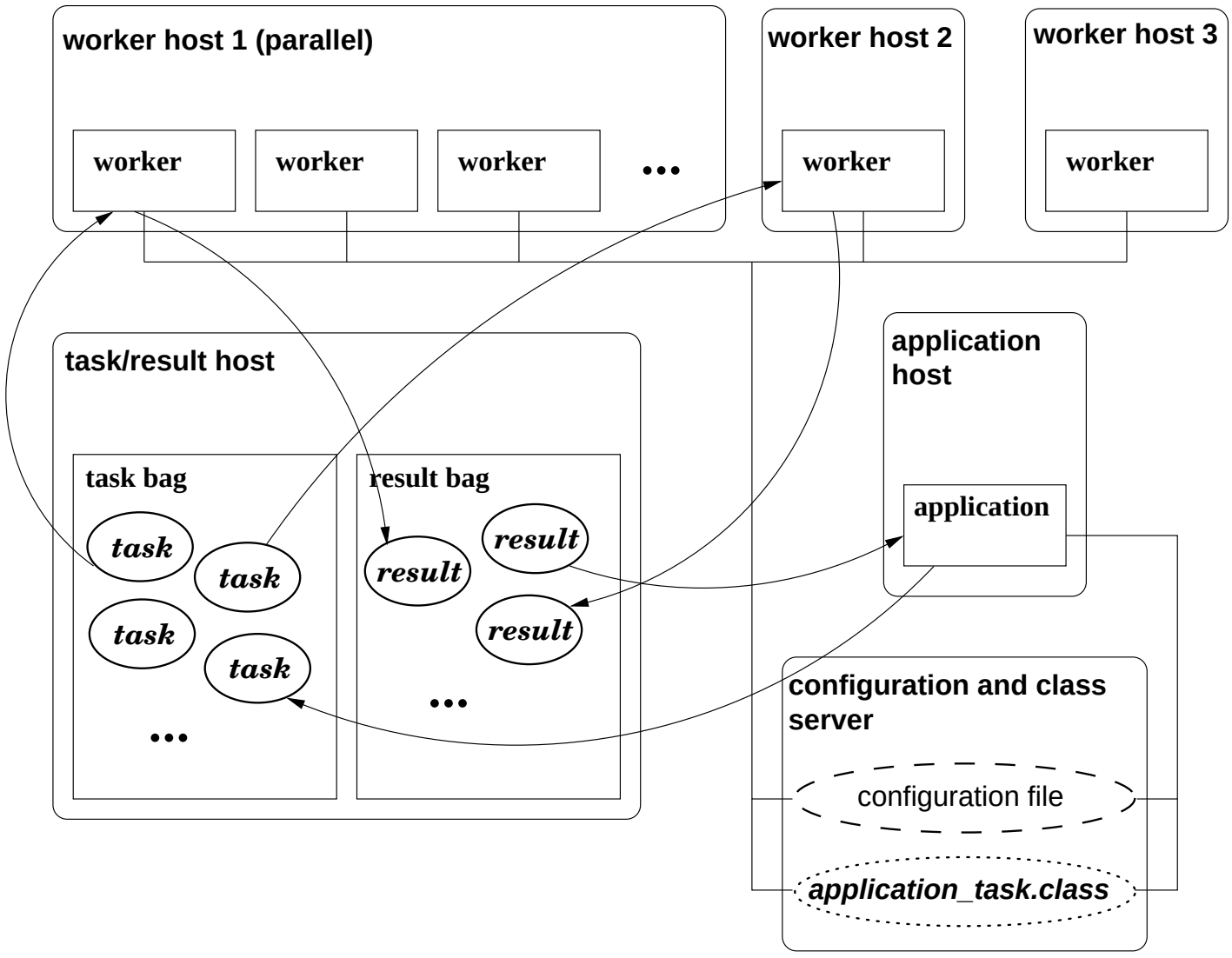
  (2) direct communication through local communication spaces

Java implementation – class diagram



- all classes extend "Communicator"
- socket connections (TCP/IP), object streams, serializable objects
- security: digitally signed jar files
- event notification in stead of polling mechanism

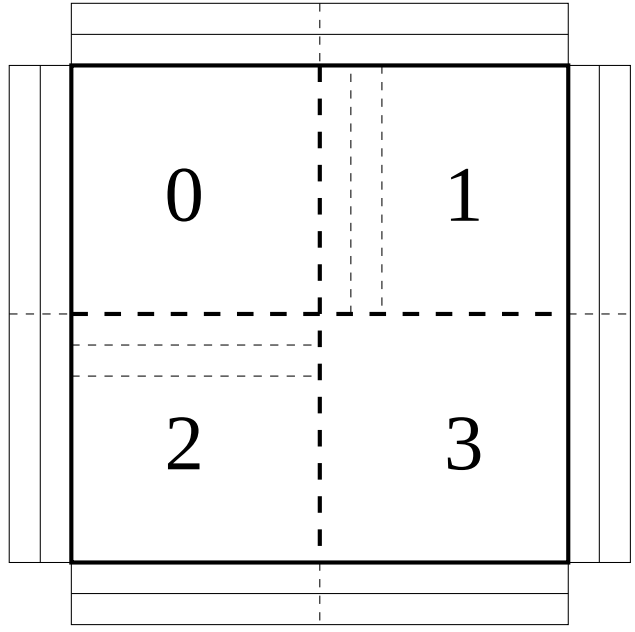
Taskspaces deployment diagram



- Runner.class < 2 kb
- task agent / reset agent

2. Parallel scaling tests – Jacobi iteration

- numerical linear algebra: Jacobi iterative method
- parallel: in every iteration neighbor–neighbor communication (exchange of boundary rows)



- constant problem size per processor (1500^2 grid points)

  $\Rightarrow$ perfect parallel scaling = constant total time

Direct communication

# of processors	# of grid points	direct communication	communi-cation bag
1	1536^2	58s	58s
2x2	3072^2	74s	87s
4x4	6144^2	83s	216s
8x8	12288^2	89s	1504s

Direct communication versus communication through a central communication bag (200 Jacobi iterations on Blue Horizon, IBM SP, SDSC).

$\Longrightarrow$ direct communication scales well (constant time)

Simple grid experiment

Blue Horizon, SDSC, San Diego, CA							
32	32	32	32	30	30	30	30
P4 Linux, CU Boulder, CO							
–	–	–	–	2	2	2	2
110s	104s	109s	106s	112s	119s	114s	116s
Total execution time							

Grid experiment (200 Jacobi iterations, 1500^2 grid points per processor). Number of processors (1 worker process per processor) and total execution times are shown.

$\Longrightarrow$ near-perfect scaling for problems with neighbor-neighbor communication over internet

High throughput grid experiment

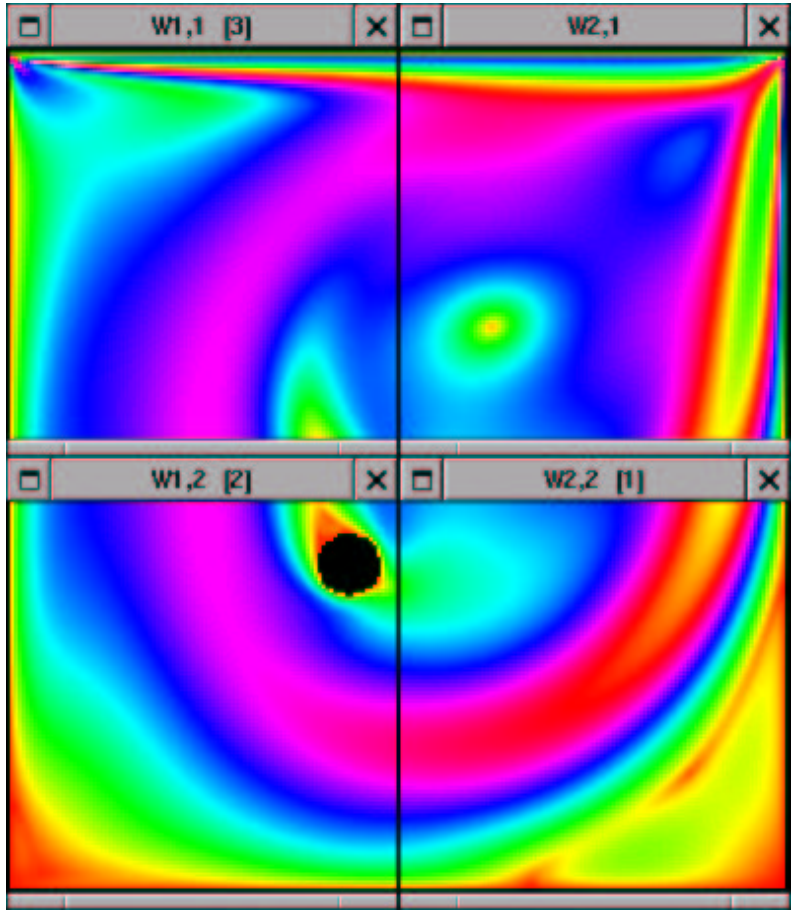
Blue Horizon, SDSC, San Diego, CA (4 workers/processor)	64	128	240
P4 Linux, CU Boulder, CO (2 workers/processor)	4	4	4
Itanium Linux, CU Boulder, CO (2 workers/processor)	4	4	4
forseti1, NCSA, Urbana, IL (1 worker/processor)	16	16	16
hermod, NCSA, Urbana, IL (1 worker/processor)	16	16	16
Total number of workers	104	168	280
Total execution time	105s	103s	101s

High throughput grid experiment (50 Jacobi iterations, 500^2 grid points per worker). Number of worker processes and total execution times are shown. Problem size is constant per worker process.

$\Longrightarrow$ Taskspaces framework: parallel scientific computing with intertask communication is scalable on the internet!

3. Other applications

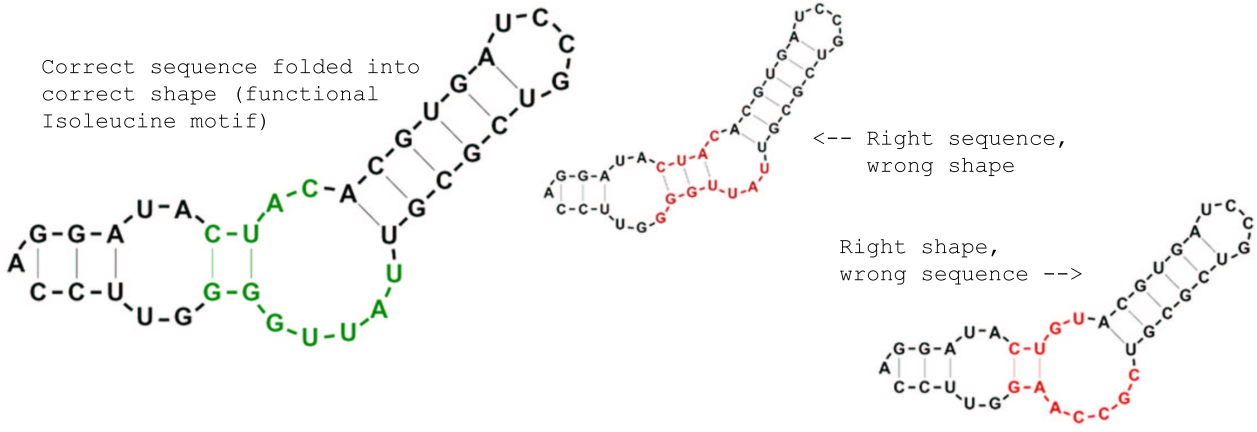
(1) engineering design collaboration



- Navier-Stokes gas dynamics, Lattice-Boltzmann
- “driven cavity”
- neighbor-neighbor communication in every step
- obstacles can be added interactively
- versatility of framework!

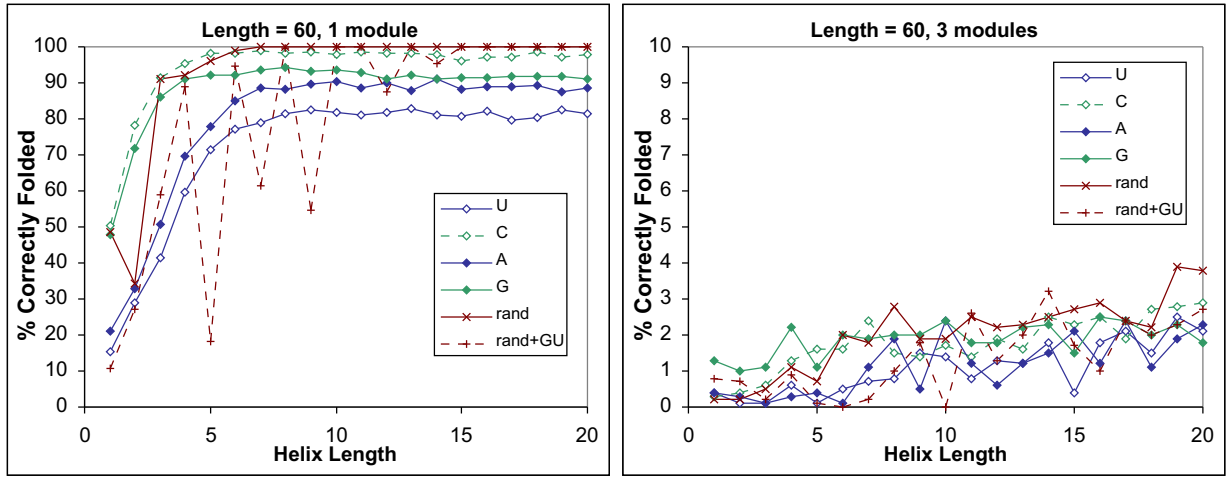
(2) Bioinformatics: RNA origin of life

with Rob Knight, MCDB, CU Boulder



- fold very large numbers ($\sim 10^9$) of random RNA sequences (length 30-500)
- look for "functional motifs" in the RNA structures that catalyze biochemical reactions
- main question: can life have originated from a small pool of random RNA molecules?

Origin of life: from random molecules!



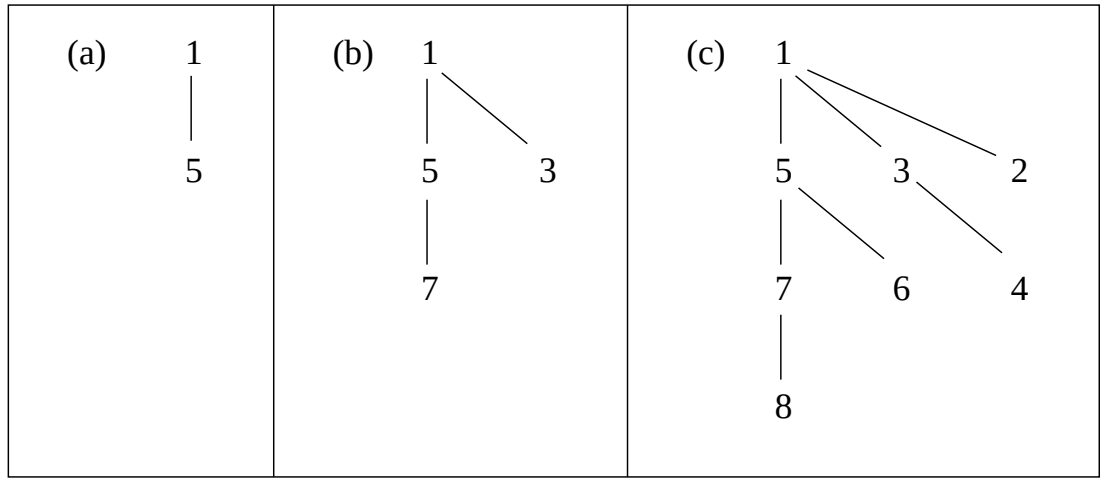
- initial approach: 46 million RNA sequences, ~ 1000 tasks, 8 hours/task on Platinum (1GHz P3 linux cluster at NCSA, 968 compute processors)
- for folding, use C-code "Vienna" compiled on workers, called from Java
- using Taskspaces, one of the missing links in the chain of life is being established!

4. Future work

- we seek collaborators and government/corporate funding to extend our framework as follows:

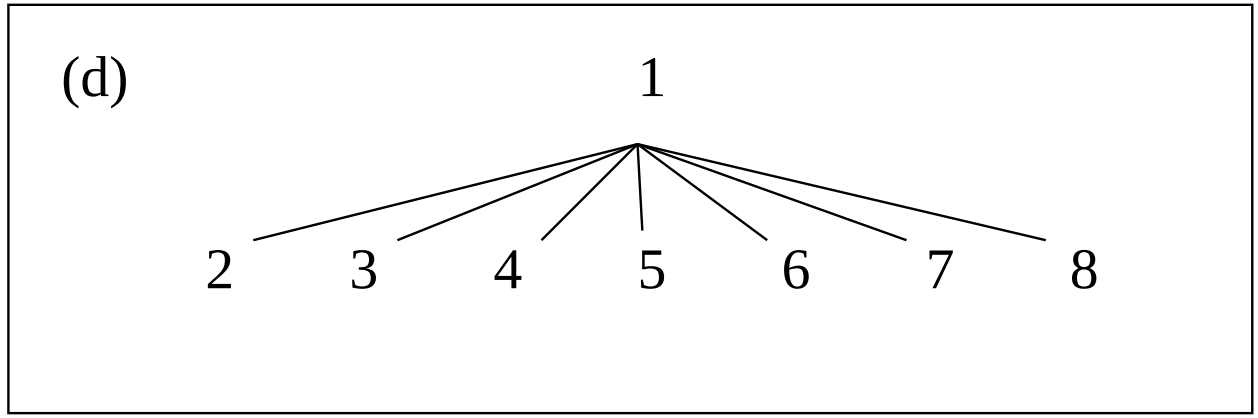
(1) Optimal collective communication trees

- Taskspaces scales for neighbor-neighbor communication
- collective communication: broadcast from one to all, reduce, . . .
- within parallel computers: optimal strategy = binomial tree

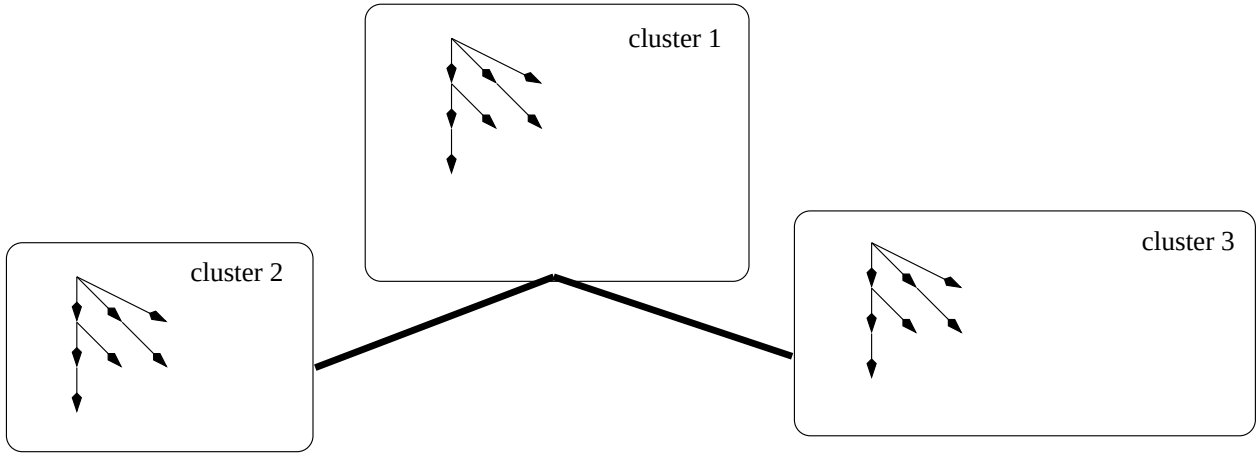


Collective communication

- high-latency/low-bandwidth connections: flat tree optimal



- multiple clusters, heterogeneous grids: optimal tree is somewhere in between

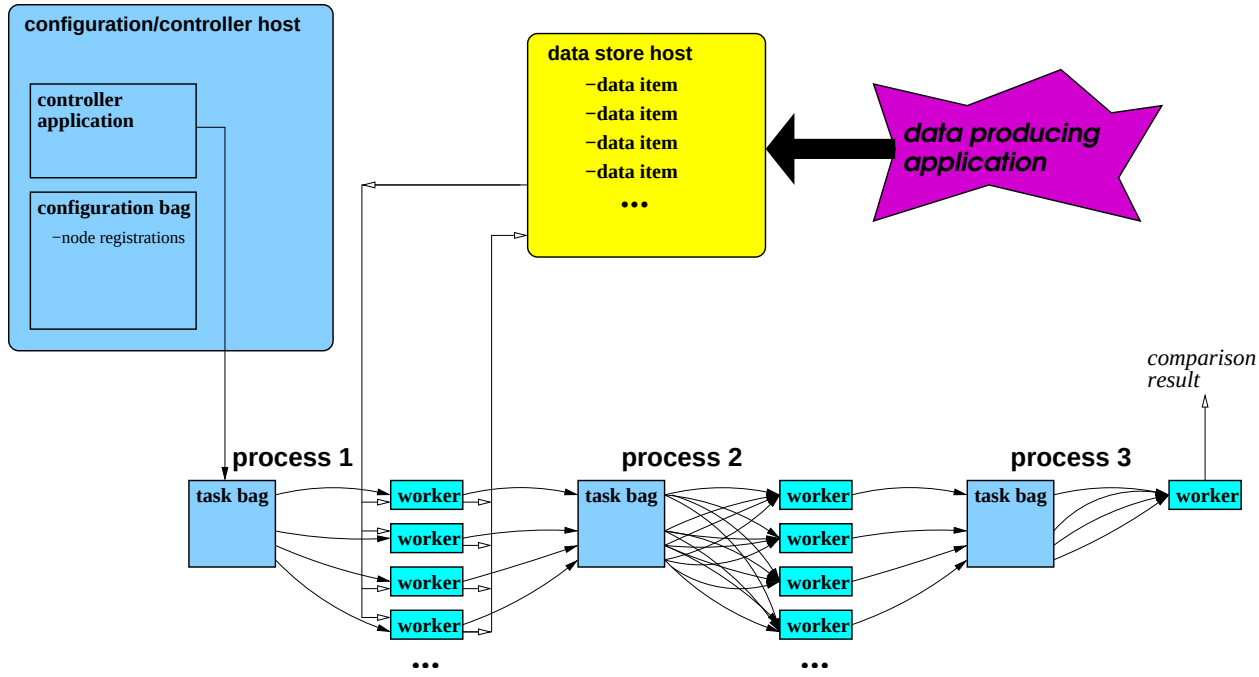


Collective communication: optimal trees

- we propose:
  - dynamically test communication time between nodes
  - grid discovers its network topology dynamically
  - determine optimal collective communication trees using clustering algorithm

  $\Rightarrow$ scalable collective communication!

(2) Self-migrating parallel workflows



- heterogeneous collection of machines
- agent sets up parallel workflow (e.g., proteomics data processing)
- upon partial or total failure (catastrophic events), parallel workflow migrates

  $\Rightarrow$ e.g., homeland security applications

A Lightweight Java Taskspaces Framework for Scientific Computing on Computational Grids

H. De Sterck[1], R.S. Markel[2], T. Pohl[3], and U. Rüde[3]

[1]Department of Applied Mathematics, University of Colorado at Boulder, USA

[2]Advansys, Inc., Boulder, CO, USA

[3]Department of Computer Science, University of Erlangen-Nürnberg, Germany

SAC 2003 - PDSN

Tuesday, March 11, 2003

(desterck@colorado.edu)