Maestro: a Self-Organizing Dataflow Framework

Kees van Reeuwijk

reeuwijk@few.vu.nl

13 June 2009
HPDC 2009

Motivation

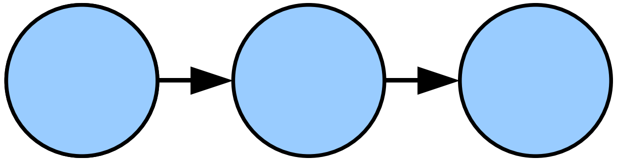
- Parallel systems are often inhomogeneous and unreliable
- Communication links are often inhomogeneous or imperfect too
- Parallelism is increasingly mainstream (multi-core, GPUs, specialized processors). Even a single consumer PC can be a heterogeneous system.
- Call it what you want: distributed system, grid, cloud, cluster...

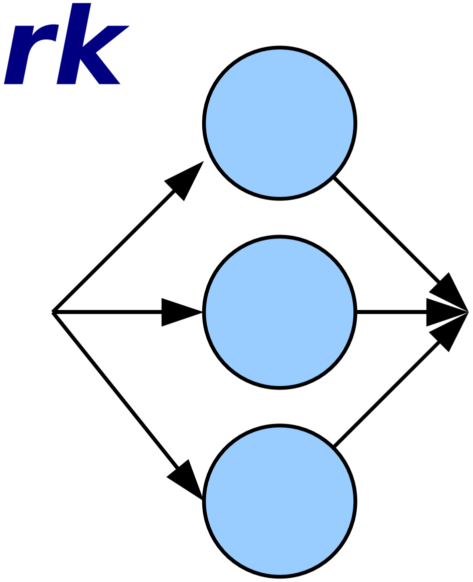
Distributed Systems Problems

- Keeping an application running (efficiently) is hard!
  - Resources come and go
  - Resources crash
  - Heterogeneous: load balance??
- Any fixed use of resources is bound to fail

Resource allocation must be dynamic and adaptive

Dataflow framework





- Jobs with one input,
  one output

```
    interface Job {
           Object run(Object in); }
```

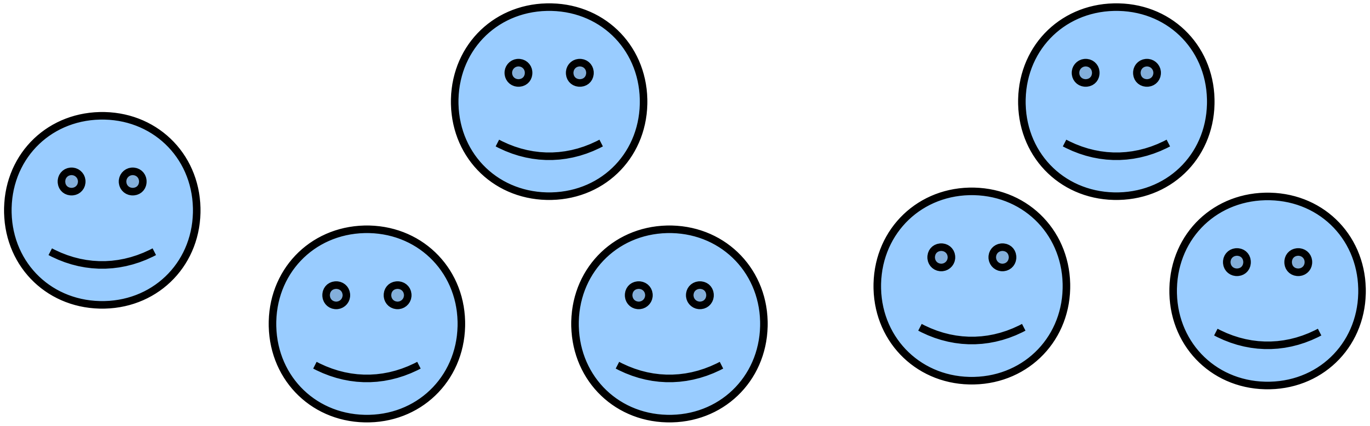
- Jobs connected in series (pipeline) or in parallel
- Nested
- Predictable performance *per job*

Maestro: self-organizing

- Nodes with special tasks are failure points/bottlenecks
- In particular central nodes (scheduler!)

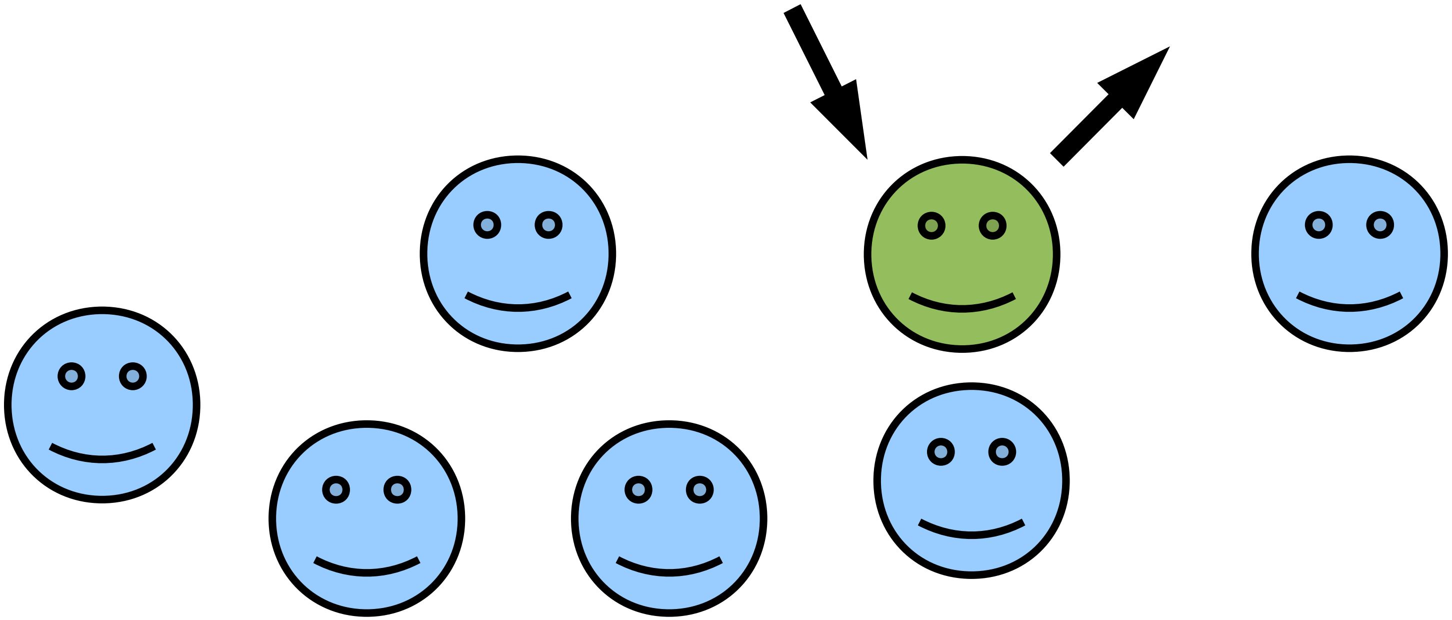
Solution: peer to peer

⇒ self organizing



Exception: work insertion

- Currently there is one exception: only one node inserts work in the system, and handles final results
- Application specific

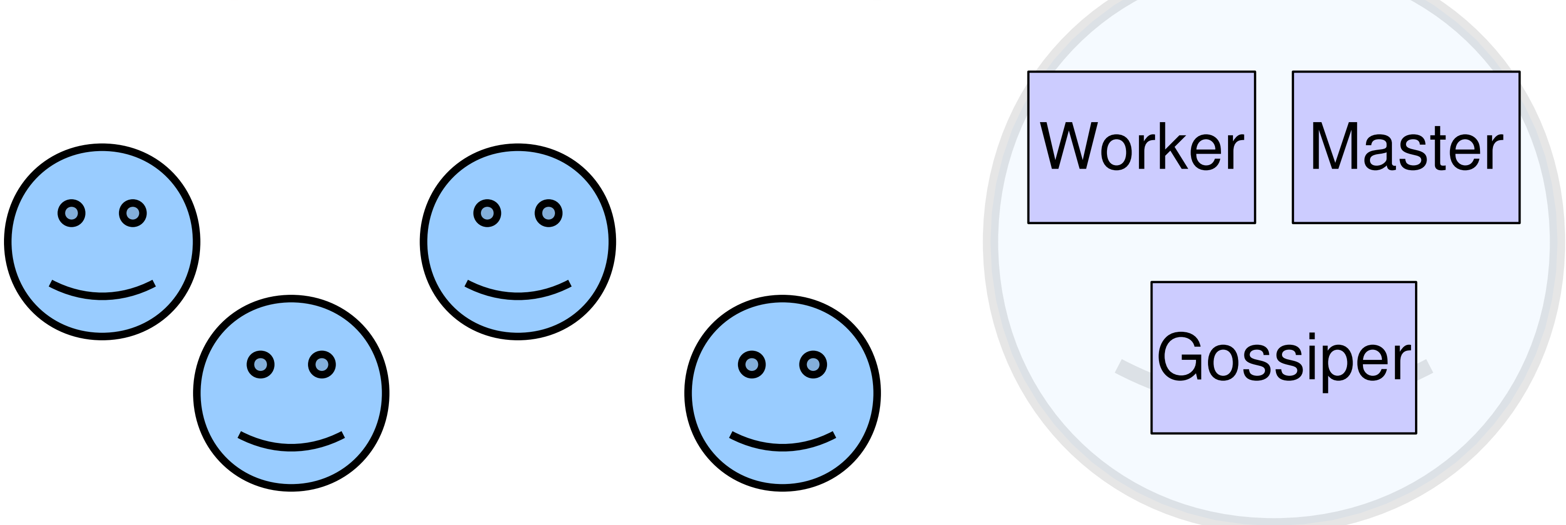


Maestro Nodes

Any number, may join and leave any time

Each node contains:

- Worker: execute jobs from queue
- Master: distribute jobs over workers
- Gossiper: exchange performance info



Scheduling policy

- Each master tries to optimize for total completion time of all remaining steps
- Measured and gossiped:
  - Worker queue & compute stats
  - Master queue stats
  - Transmission time (not gossiped)
- Regulars are informed ASAP
- Efficient nodes are favored

Learning strategy

Emergent behavior: the system learns an efficient schedule:
reenforcement learning

Consequences:

- In a homogeneous system the local node is favored
- New nodes should start with optimistic estimates

Limited commitment

Every worker should have one job waiting in its queue: no more, no less

- Limits commitment to one node, but reduces idle time
- Gives opportunities to less attractive nodes

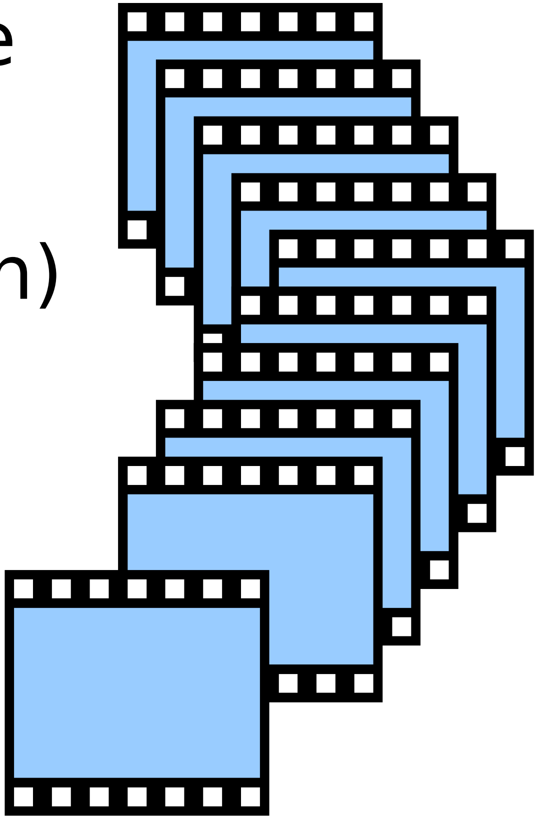
Implemented on Ibis

- A framework for distributed computing
- Based on Java (portable!)

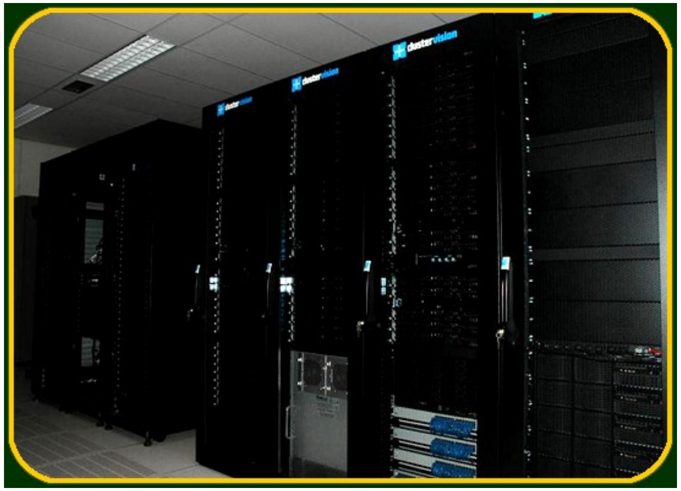
- Provides message passing, serialization (IPL layer)
- Join-Elect-Leave support (malleability)
- Robustness is central
  - Detect failed nodes
  - Circumvent NATs, firewalls, etc.
  - Handle multiple NICs (multi-homing)

Benchmark

- Operations on video frames
  1. Generate 720x576 frame
  2. Scale to 1440x1152
  3. Sharpen (3x3 convolution)
  4. Compress (JPEG)
  5. Discard



Testbed



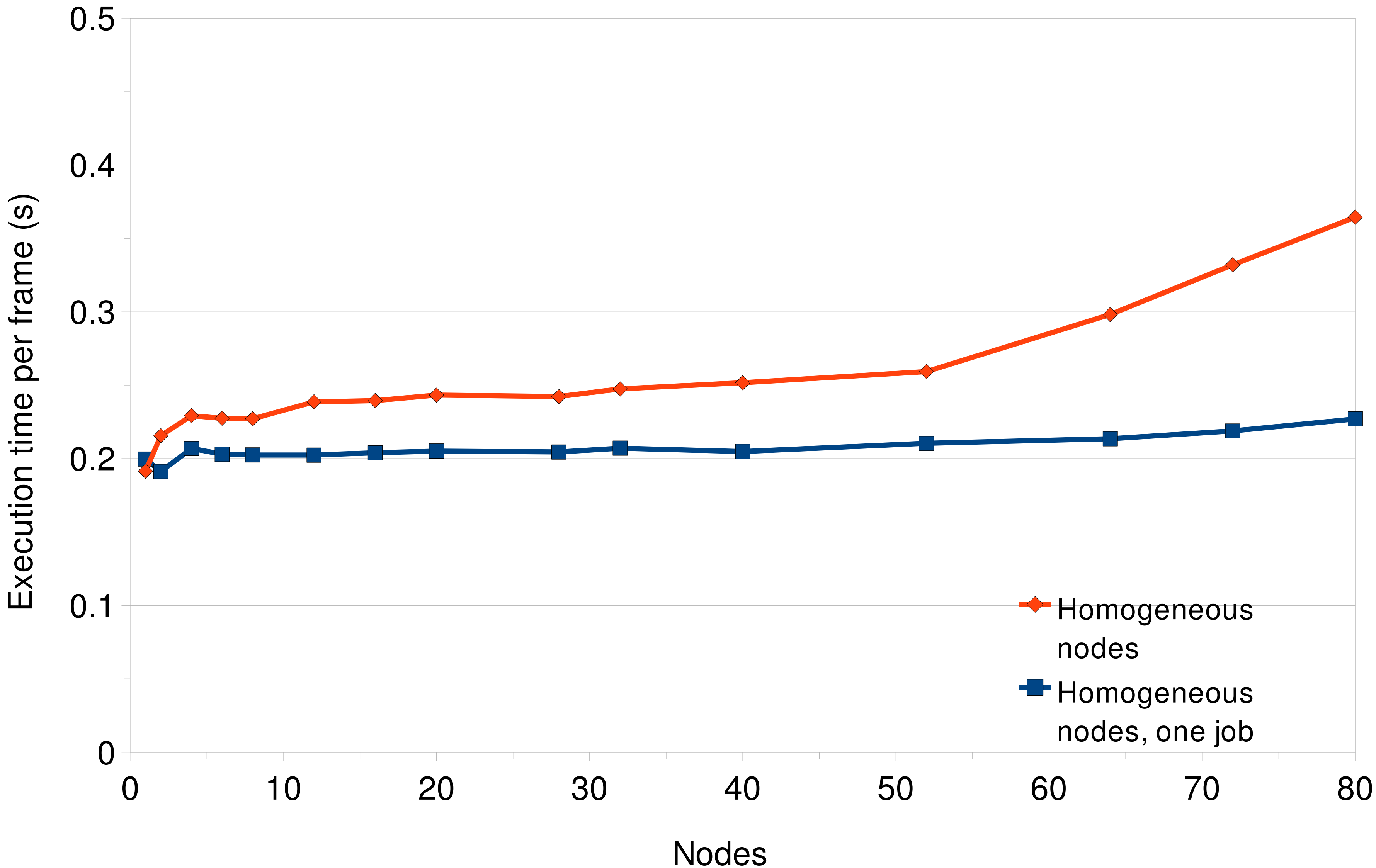
VU cluster of the DAS3:

- 85 nodes:
  - 2x dual-core 2.4 GHz AMD Opteron
  - 4 GB memory
- Myrinet 10G interconnect
- In total there are 5 clusters with similar specs throughout the Netherlands

Node configurations

- Homogeneous
  - We expect:
    - Work is evenly divided over the nodes
    - All five stages of the video processing on the same node
- All steps in one job
  - We expect:
    - Work is evenly divided
  - Maestro is just used as master/worker

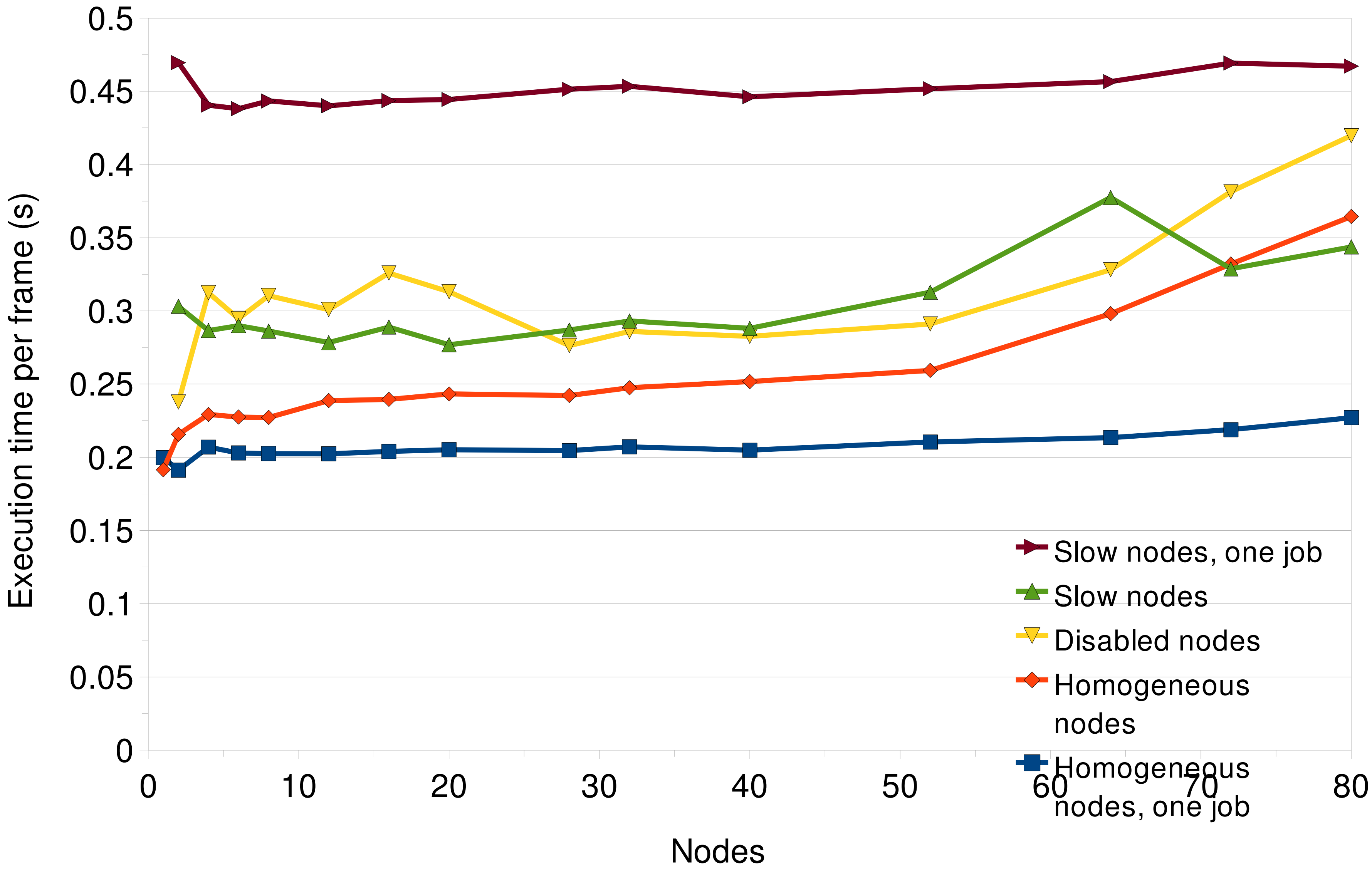
Homogeneous results



Heterogeneous configurations

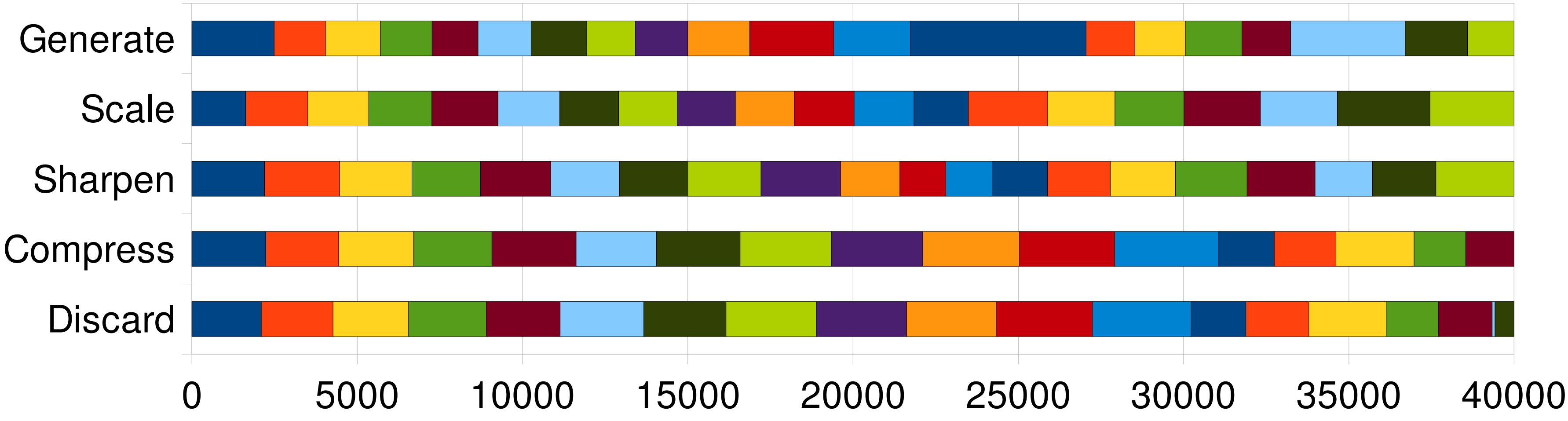
- Half no scaling, half no sharpening
  - Now forced to `zigzag'
- Slow scaling, slow sharpening
  - At least the `zigzag'
- One job, slow scaling, sharpening
  - Slow computation unavoidable

All results

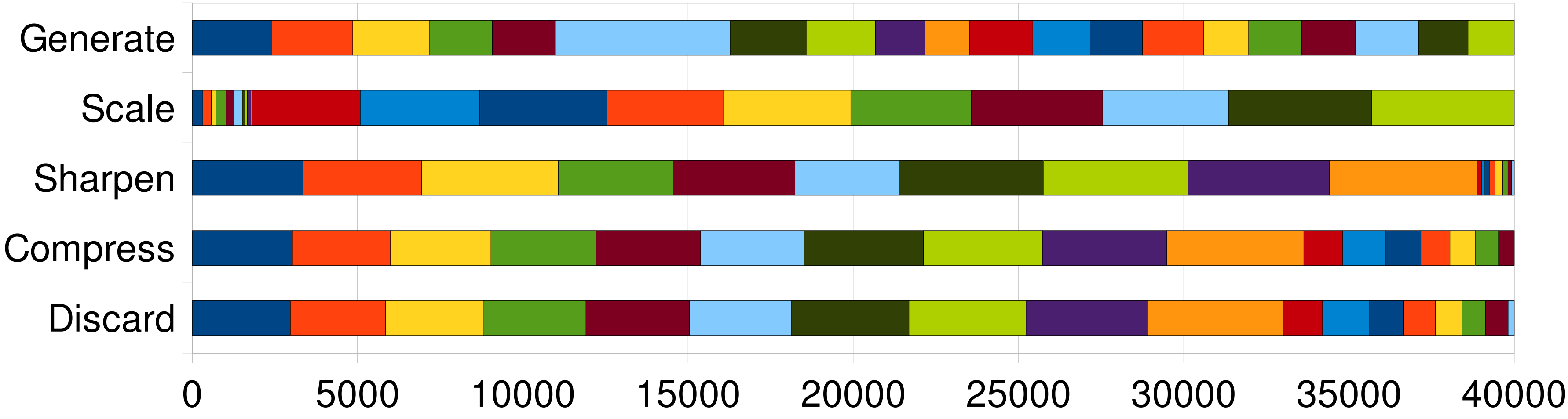

0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
Execution time per frame (s)
0
10
20
30
40
50
60
70
80
Nodes
Slow nodes, one job
Slow nodes
Disabled nodes
Homogeneous nodes
Homogeneous nodes, one job

Work distribution

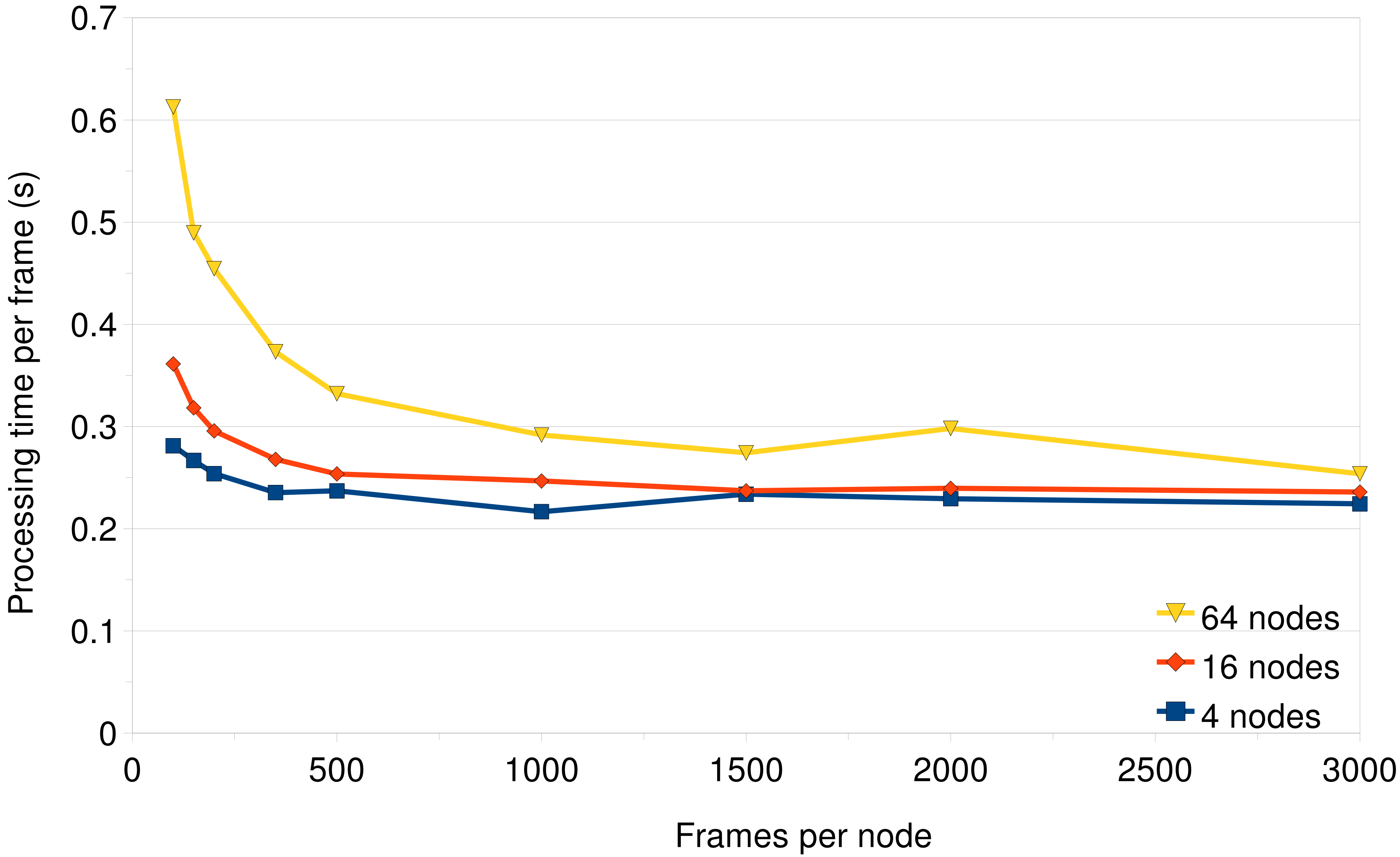
20 homogeneous nodes



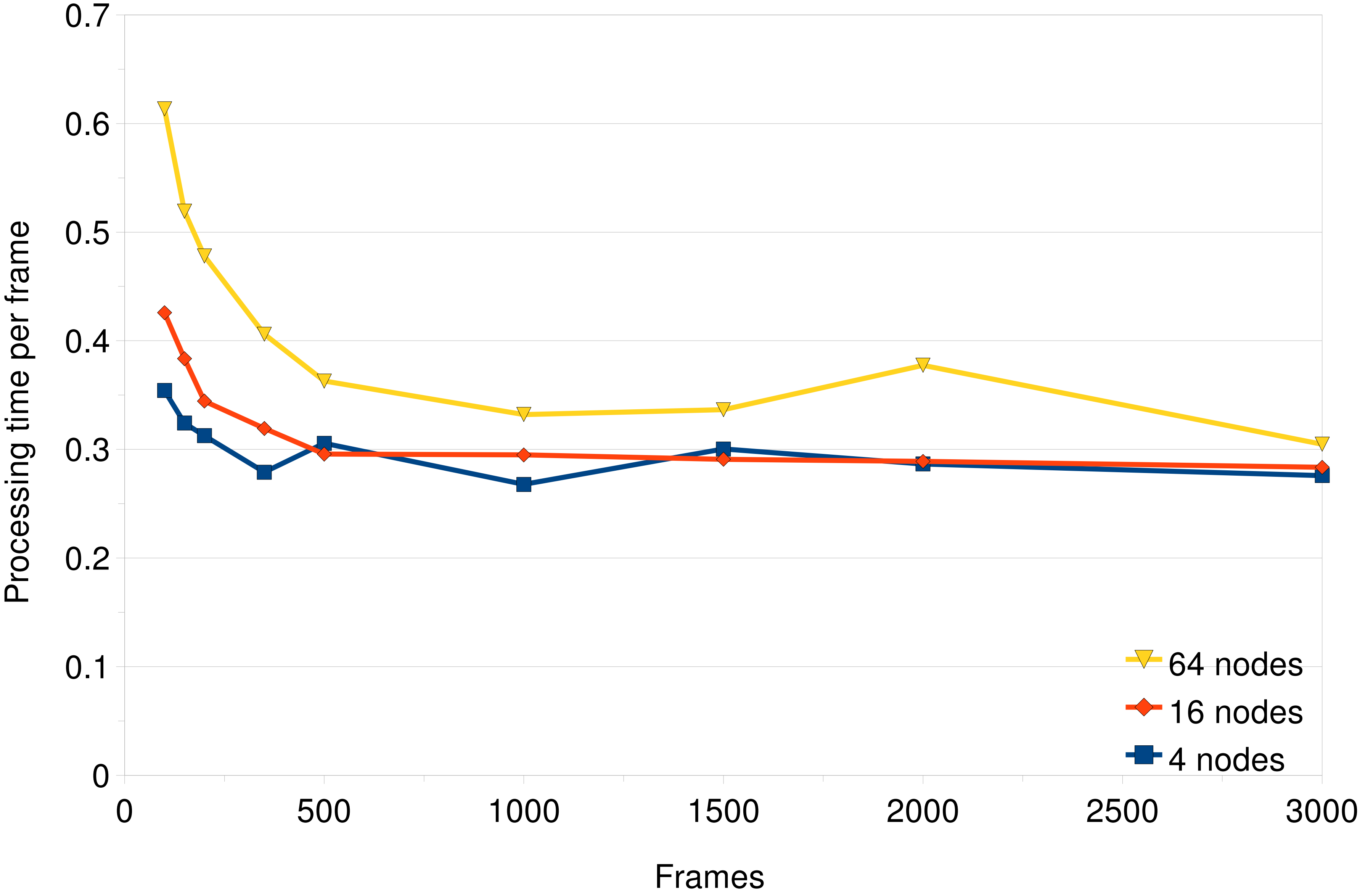
10 nodes with slow scaling, 10 with slow sharpening



Learning: homogeneous

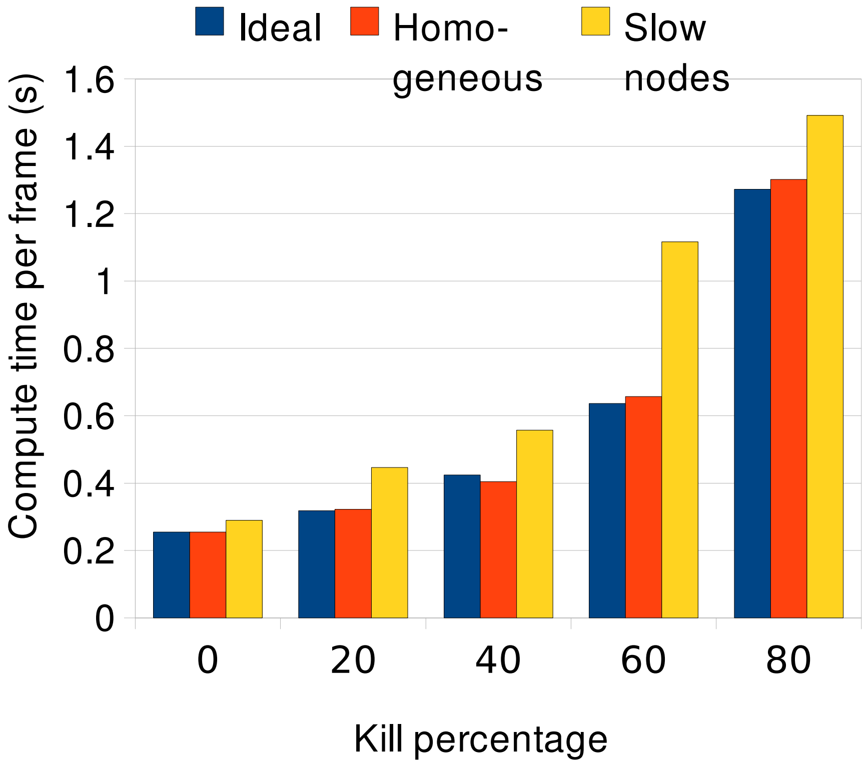


Learning: slow nodes



Fault tolerance

- We start a run on 30 nodes
- After a few seconds kill some nodes
- Ideally, the rest of the nodes should take over the work



- All masters restart any work that was lost on the dead nodes
- Retry outstanding frames

Conclusions & future work

Conclusions

- Self-organization of a data-flow computation works
- Can exploit strong points of non-homogeneous systems
- Extremely robust

Future work

- Integrate with divide & conquer
- Scalability

Questions?

?

www.cs.vu.nl/ibis