

# A Comparison of Local and Gang Scheduling on a Beowulf Cluster

Peter Strazdins\*  
and John Uhlmann,

Department of Computer Science,  
The Australian National University

Cluster 2004, 21 September 2004

<http://cs.anu.edu.au/~Peter.Strazdins/seminars#SchedBeowulf>

# 1 Outline

- introductory concepts for gang and local scheduling ⇒
- related work ⇒  
and contributions of this work ⇒
- the Bunyip Beowulf cluster and LAM/MPI under Linux 2.4 ⇒
- the SCore Cluster Management System ⇒
- experimental setup for the comparison ⇒
- results for matrix multiply and Linpack benchmarks ⇒
- an optimistic performance model for local scheduling ⇒
- experimental evaluation of the model ⇒
- conclusions ⇒  
and future work ⇒

## 2 Gang and Local Scheduling; Introductory Concepts

- run time-shared multiple parallel jobs on a series of processors
  - e.g. run 4-process parallel jobs J0 and J1 on 4 processors

	...	...	...	...	...	...	...	...
	J1	J1	J1	J1	J0	J1	J1	J0
	J0	J0	J0	J0	J1	J0	J0	J1
	J1	J1	J1	J1	J1	J1	J0	J1
	J0	J0	J0	J0	J0	J0	J1	J0
↑ time	$p_0$	$p_1$	$p_2$	$p_3$	$p_0$	$p_1$	$p_2$	$p_3$

scheduling:	gang	local
timeslices:	0.2s	0.01s
messaging:	busy-wait	yield if not ready
implementation:	complex	simple

- desire to run small (interactive) jobs fairly with larger (production) jobs with high efficiency (high throughput of jobs)
  - wide belief that all of a job's processes must be scheduled together
    - o.w. 'thrashing' (context switching & cache pollution) may occur
  - but can't overlap communication of 1 job with computation of another

### 3 Related Work

- large body of research related to gang scheduling (or similar);
  - incorporated into the ParPar and SCore cluster management systems
  - Dror Feitelson 99: “Scheduling on [clusters] is essentially the same as on commercial MPP systems”
- some research on potential benefits of local scheduling (or similar), but little actual evaluation on real parallel workloads
- belief that gang (or similar) scheduling is necessary seems to be based on few direct comparisons:
  - Feitelson & Rudolph 92 (JPDP 16(4)): synthetic (computation loop / barrier) program on an early NUMA architecture
  - Atsushi Hori et al (PDTTPA'98): direct comparison on NAS || Benchmarks
    - local scheduling with variable spin-wait time was never better than gang on a 32-node Pentium Pro/Myrinet cluster under SCore
  - Feitelson & Wiseman (IEEE TPDS 06/03): Paired Gang Scheduling
    - proposed as compromise between gang & local scheduling
    - evaluation on synthetic workloads; no direct comparison with local

## 4 Contributions of this Work

- a direct comparison of gang and local scheduling on a Beowulf-style Linux cluster computer
  - using -synthetic applications, importantly having:
    - significant memory footprint
    - 'interesting' communication patterns
  - sub-linear slowdowns observed for both policies!
  - demonstrates a decisive advantage of local scheduling, even with no supporting infrastructure added
- proposes a performance model for 'ideal' local scheduling, and evaluates this under the above applications

## 5 The Bunyip Beowulf Cluster and LAM/MPI under Linux 2.4

- 'Bunyip' is an monster in Australian mythology
- won Gordon-Bell Award for Price/Performance in 2000
- Bunyip consists of 4 groups of 24 of 550 MHz dual Pentium III nodes
  - with shared 256 KB direct-mapped second-level cache
- each node has three 100Mb NICs
  - can communicate with 3 other nodes simultaneously
- two processes of a parallel job are normally spawned on each node
- under LAM MPI 6.3.2 / Linux 2.4:
  - inter-node communication via TCP/IP transport
    - 'small' ( $\leq 64\text{KB}$ ) messages are non-blocking to the sender
    - a process will yield upon message receive (if not arrived yet)
    - startup cost is  $\alpha = 80\mu\text{s}$ ; cost per word is  $\beta = .85\mu\text{s}$
  - intra-node communication via `sysv` transport
    - $\alpha = 42\mu\text{s}$ ,  $\beta = .10\mu\text{s}$
  - context switch overhead measured at  $\alpha_c = 2.5\mu\text{s}$

## 6 The SCore Cluster Management System

- SCore 5.4.0 has a buddy-based gang scheduler with centralised control
  - nodes are time-shared in a coordinated fashion,
  - global synchronizations and flushing of the network between time slices
- inter-node communication via lightweight PM/Ethernet transport
  - MPI implementation is based on MPICH 1.4.0
  - a process 'busy-waits' upon message receive (if not arrived yet)
  - $\alpha = 70\mu\text{s}, \beta = .82\mu\text{s}$ 
    - thus, single MPI jobs may run faster under SCore
- was installed on 16 nodes of the Bunyip
  - requires kernel patches (PM/Ethernet drivers)
  - otherwise SCore operates entirely in user-space
- SCore is widely deployed, and reported to be highly popular amongst users

## 7 Experimental Setup

- used two benchmark programs, matrix multiply and Linpack
  - only MPI calls used are simple send and receive calls
  - have minimal I/O: only process 0 produced output messages
- $j \geq 1$  identical || jobs are simult. spawned over the same subset of nodes
  - both LAM/Linux & Score gave fair individual service to each job
  - overall throughput captured by elapsed execution time for all jobs,  $t_j$
  - the job slowdown metric reduces to  $s_j = \frac{t_j}{t_1}$
- the memory footprint of each job kept at  $\approx 6\%$  of available memory
- $p = PQ$  parallel processes run over a  $P \times Q$  logical grid ( $Q = P, 2P$ )

- e.g. for  $p = 32$  over 16 nodes:

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

- on  $N \times N$  matrices, both programs have  $\Theta\left(\frac{N^3}{PQ}\right)$  computation cost and  $\Theta\left(\frac{N^2}{Q} + \frac{N^2}{P}\right)$  communication volume costs
  - Linpack is fine grained: has  $\Theta(N)$  communication startup costs

## 8 Results for the Two Scheduling Policies

matrix multiply:	$p$	$N$	single job			$s_j, j$ simultaneous jobs			
			%CPU	%Mem	Time (s)	2	4	6	8
local	2	1000	99	5.2	3.7	1.81	3.45	5.10	6.75
(LAM/	8	2000	71	5.2	9.6	1.81	3.44	5.05	6.62
Linux)	32	4000	64	5.4	20.9	1.83	3.64	5.24	6.96
gang	2	1000	100	5.1	5.0	1.80	3.52	5.28	7.02
(Score)	8	2000	100	5.6	10.3	1.89	3.70	5.57	7.42
	32	4000	100	5.6	20.4	1.95	3.88	5.84	7.88

Linpack:

local	2	1500	72	5.2	6.8	1.85	3.50	5.30	6.91
(LAM/	8	3000	40	5.8	24.6	1.62	2.65	3.66	4.49
Linux)	32	6000	26	5.8	61.8	1.58	2.41	3.24	3.95
gang	2	1500	100	5.0	7.5	1.86	3.66	5.50	7.35
(Score)	8	3000	100	5.6	23.1	1.95	3.89	5.82	7.85
	32	6000	100	5.6	54.4	1.98	3.94	5.93	7.99

- SCore improved slightly on the (ideal) sub-linear slowdown,  $s_j = j$
- for  $p \geq 2, j \geq 2$ , local scheduling has lower  $t_j$ ,  
up to 80% faster for Linpack at  $j = 8$  and  $p = 32$

## 9 An Optimistic Performance Model for Local Scheduling

- given a single job's execution time  $t_1(\alpha, \beta)$ , the ideal execution time for  $j$  simultaneous jobs is:

$$t_j = \max(t_1(\alpha, \beta), jt_1(\alpha_c + \alpha_{CPU}, \beta_{CPU}))$$

where  $\alpha_{CPU} + \beta_{CPU}n$  is the amount of *process time* spent in transmitting a message of length  $n$

- ping-pong benchmark on Bunyip (LAM/Linux) has yielded:

$$\alpha_{CPU} = 28\mu\text{s}, \beta_{CPU} = .14\mu\text{s}$$

- has 2 assumptions:

A1:  $j$  is large enough that there is always a runnable process  
(the term  $jt_1(\alpha_c + \alpha_{CPU}, \beta_{CPU})$  corresponds to 100% CPU utilization)

A2: cache pollution due to process switching may be neglected

- have developed detailed single job execution time models:

- e.g. matrix multiply:  $t_1 = \frac{2N^3}{PQ}\gamma_3 + \frac{N^2}{PQ}(P + Q - 2)\beta$

- $\gamma_3$  is the cost per (BLAS Level 3) floating point operation

- only cover the main computation of the benchmark, not the whole job

## 10 Experimental Evaluation of the Optimistic Model

$p = 32$	matrix multiply, $N = 4000$					Linpack, $N = 6000$				
	$t_1$ (s)	$s_2$	$s_4$	$s_6$	$s_8$	$t_1$ (s)	$s_2$	$s_4$	$s_6$	$s_8$
expt (job):	20.9	1.83	3.59	5.24	6.96	61.8	1.58	2.41	3.24	3.95
from %CPU:		1.86	3.49	5.15	6.87		1.56	2.27	2.96	3.66
% error (A2)		0%	3%	2%	1%		1%	6%	9%	7%
expt (main):	18.8	1.92	3.63	5.26	6.89	58.6	1.56	2.35	3.09	3.83
model (main):	17.7	1.12	2.25	3.38	4.50	56.2	1.00	1.69	2.54	3.38
% error (total)		42%	38%	36%	35%		36%	28%	18%	12%
% error (A1)		42%	35%	34%	34%		35%	22%	9%	5%

- the %CPU utilization indicates the actual extent of inter-job overlap computation with communication (but not cache pollution (A2))
- comparing the experiment to model (over main computation) indicates the total error
- final row indicates how close Linux's local scheduling is to the ideal
  - close for Linpack; still unclear why not closer for matrix multiply
- model predicts Linpack has  $s_8 = 5.4$  for 2GHz CPUs & 1Gb network

## 11 Conclusions

- local scheduling can significantly out-perform (strict) gang scheduling on some applications on Beowulf-style cluster computers
  - the ability to overlap inter-job computation and communication leads to sub-linear slowdowns
  - the application need not be coarse-grained; indeed, a mixture of large and small messages may be helpful
  - the effect of (direct) process switch overheads were negligible; resulting overhead from cache pollution is more significant
  - near-ideal scheduling behaviour observed under LAM/Linux for Linpack
- both (cache) memory usage and communication patterns can have an important effect on overall job throughput
- how essential is gang scheduling, given its complexity of implementation?

## 12 Future Work

- important issues beyond the scope of this work:
  - efficient job packing (space-sharing)
  - (effectively) limiting the active (i.e. ready-to-run) jobs based on total memory utilizationmust be considered in real cluster management systems
- future work includes:
  - more comprehensive evaluations
  - instrumenting the Linux kernel for a better understanding of local scheduling
  - optimizing (near-) local scheduling on a 'cluster-aware' kernel
    - only the kernel can have complete knowledge of the system
  - alternatively, enhancing gang scheduling to be able to overlap inter-job computation and communication
    - extend paired gang scheduling to 'grouped' gang scheduling
    - Atsushi Hori: preempt a job if most of its processes are waiting on communication