Flexible Coscheduling

Eitan Frachtenberg[1,2], Dror Feitelson[2], Fabrizio Petrini[1], Juan Fernandez[1]

[1] CCS-3 Modeling, Algorithms, and Informatics Group

Computer and Computational Sciences (CCS) Division

Los Alamos National Laboratory

`{eitanf,fabrizio,juanf}@lanl.gov`

[2] School of Computer Science and Engineering

Hebrew University, Jerusalem, Israel

`feit@cs.huji.ac.il`

IPDPS 2003

Outline

- **Parallel job scheduling**
  - Where we are
  - Recent challenges and opportunities

Outline

- **Parallel job scheduling**
  - Where we are
  - Recent challenges and opportunities
- **Flexible coscheduling**
  - New job scheduling method
  - Various kinds of applications and workloads

Outline

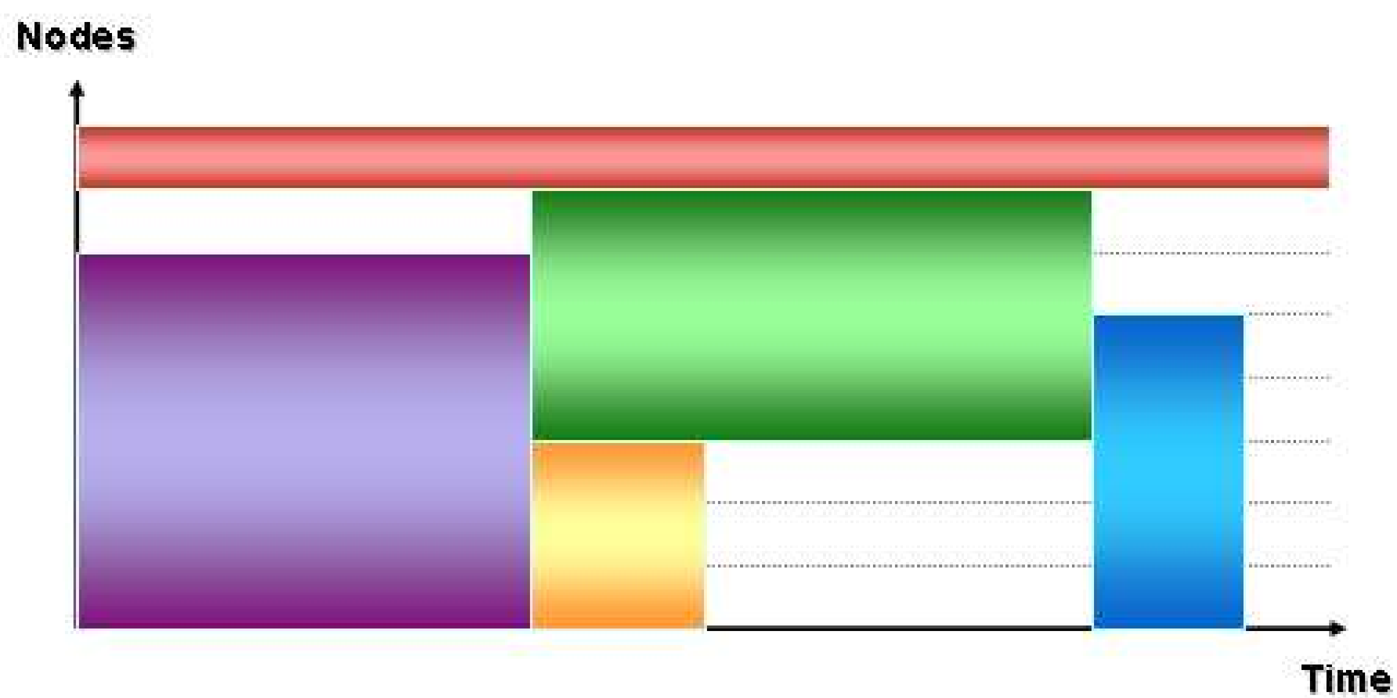
- **Parallel job scheduling**
  - Where we are
  - Recent challenges and opportunities
- **Flexible coscheduling**
  - New job scheduling method
  - Various kinds of applications and workloads
- **Performance**
  - Synthetic tests
  - Real applications
  - Dynamic workloads

Parallel Job Scheduling - Space Slicing

- Processors are divided to partitions
- Various implementations (CM-5, SP2, Cray T3D, BG/L)
- Each job runs to completion in its dedicated partition
- Batch scheduling - no preemption

Parallel Job Scheduling - Space Slicing

- Processors are divided to partitions
- Various implementations (CM-5, SP2, Cray T3D, BG/L)
- Each job runs to completion in its dedicated partition
- Batch scheduling - no preemption



Parallel Job Scheduling - Time Slicing

- Multiprogramming in a parallel machine
- Improve utilization, response time, interactivity

Parallel Job Scheduling - Time Slicing

▷ Multiprogramming in a parallel machine

▷ Improve utilization, response time, interactivity

Challenges:

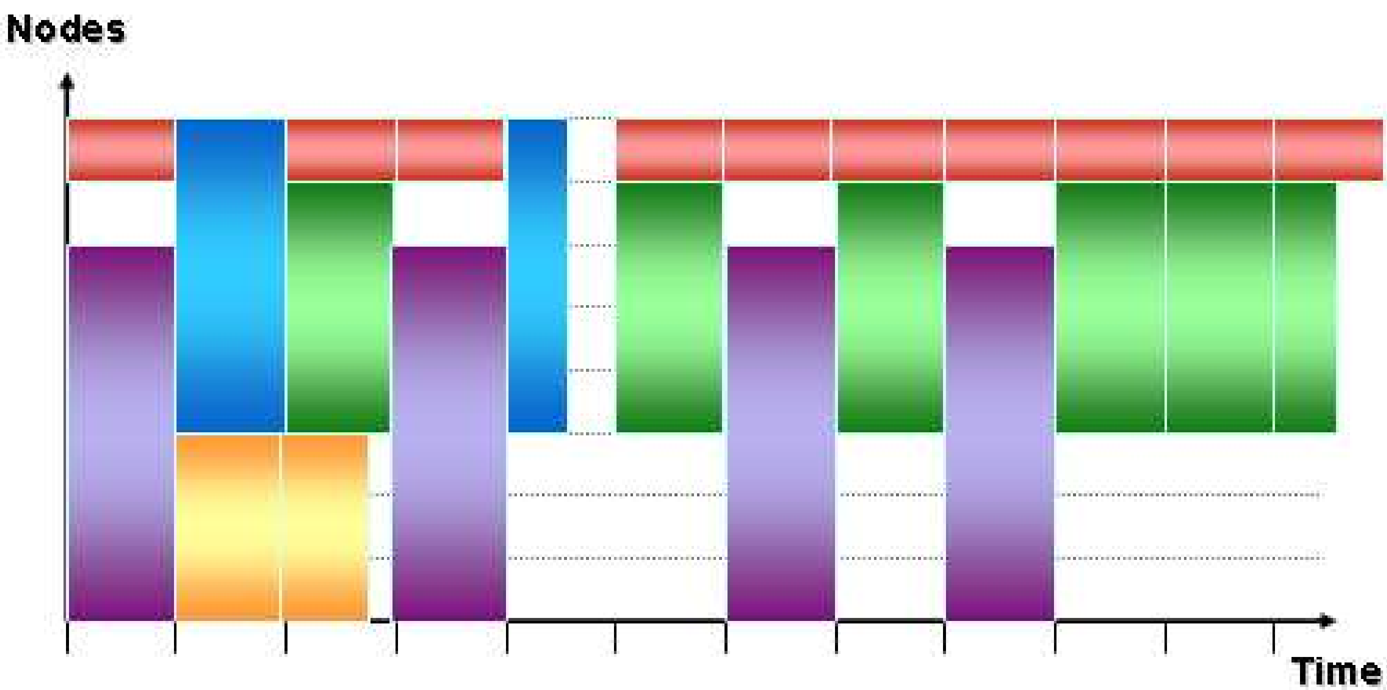
- Scalability: machines and clusters are growing
- Overhead, cache, and memory pressure
- Flexibility: various jobs and workloads:
  - Cooperating processes need to be scheduled together
  - Load imbalance

Explicit Coscheduling

- Gang Scheduling (GS): coordinated context switching
- Context switch incurs overhead and cache pressure
- Scalability issues with global context switch

Explicit Coscheduling

- Gang Scheduling (GS): coordinated context switching
- Context switch incurs overhead and cache pressure
- Scalability issues with global context switch

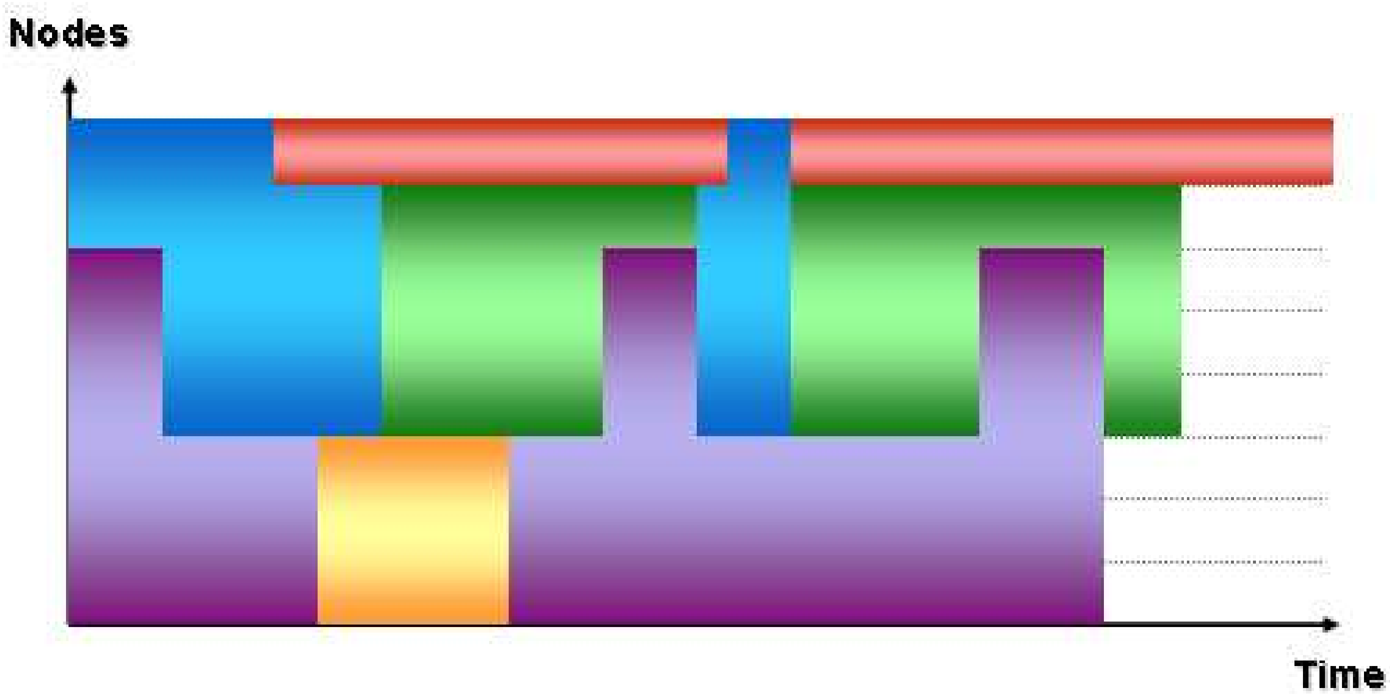


Implicit Coscheduling

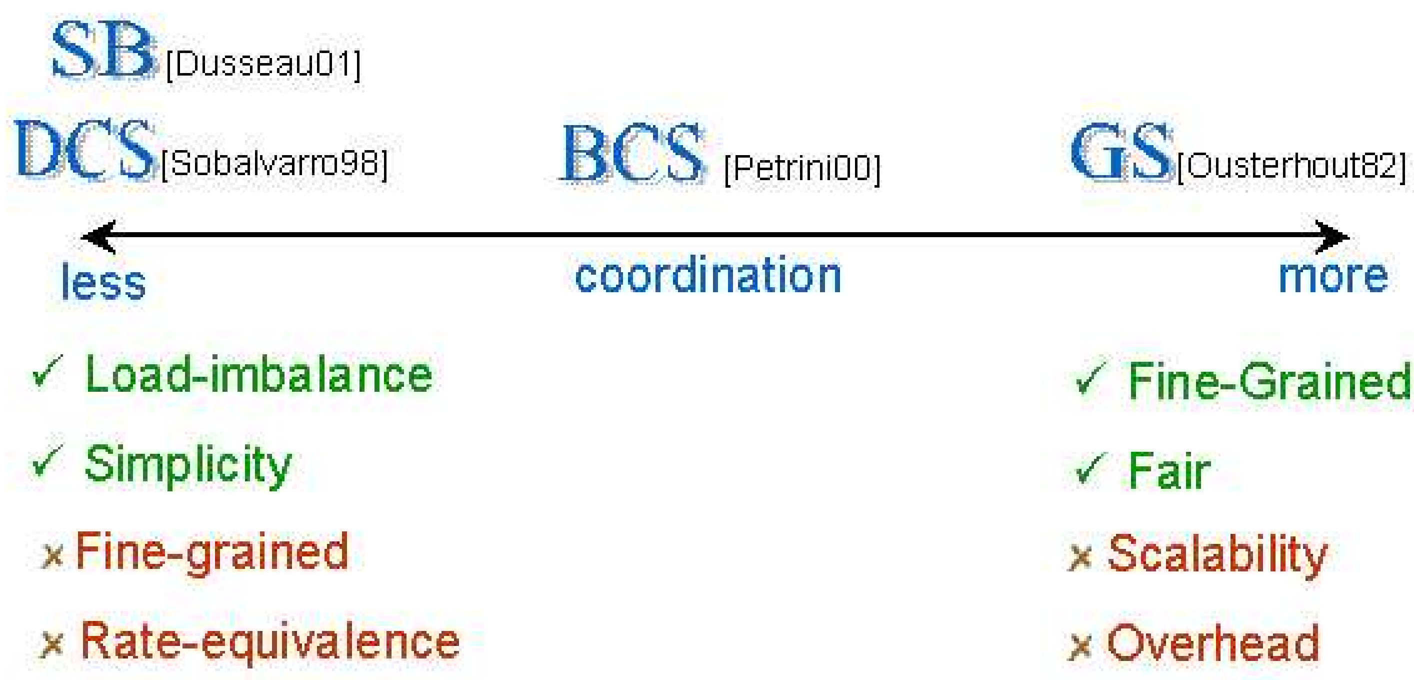
- Various methods: DCS, SB, PBT, ICS,...
- Use only local information for coordination
- Good for load-imbalance and utilization
- So-so for fine-grained or rate-equivalent jobs

Implicit Coscheduling

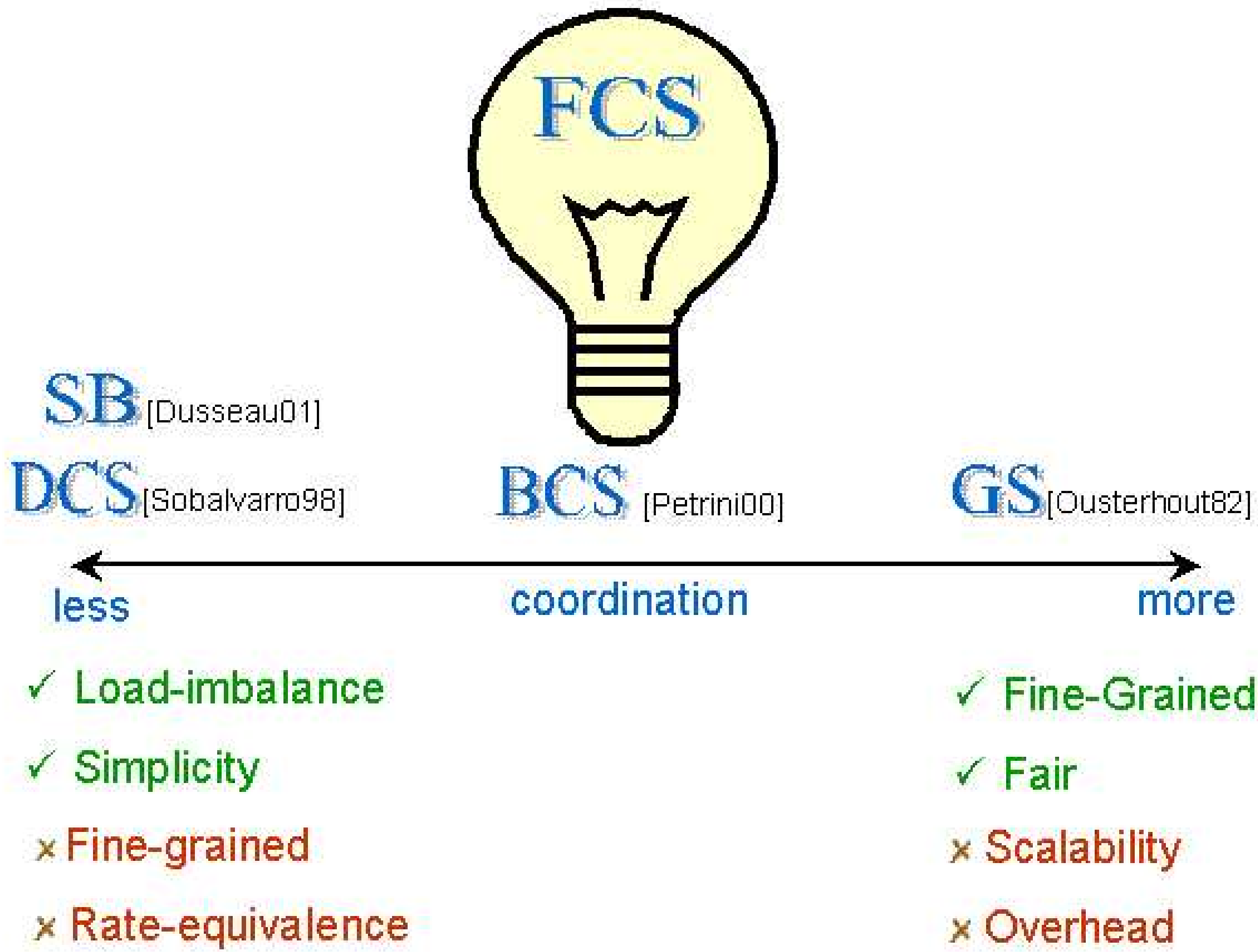
- Various methods: DCS, SB, PBT, ICS,...
- Use only local information for coordination
- Good for load-imbalance and utilization
- So-so for fine-grained or rate-equivalent jobs



Time-Slicing Scheduling



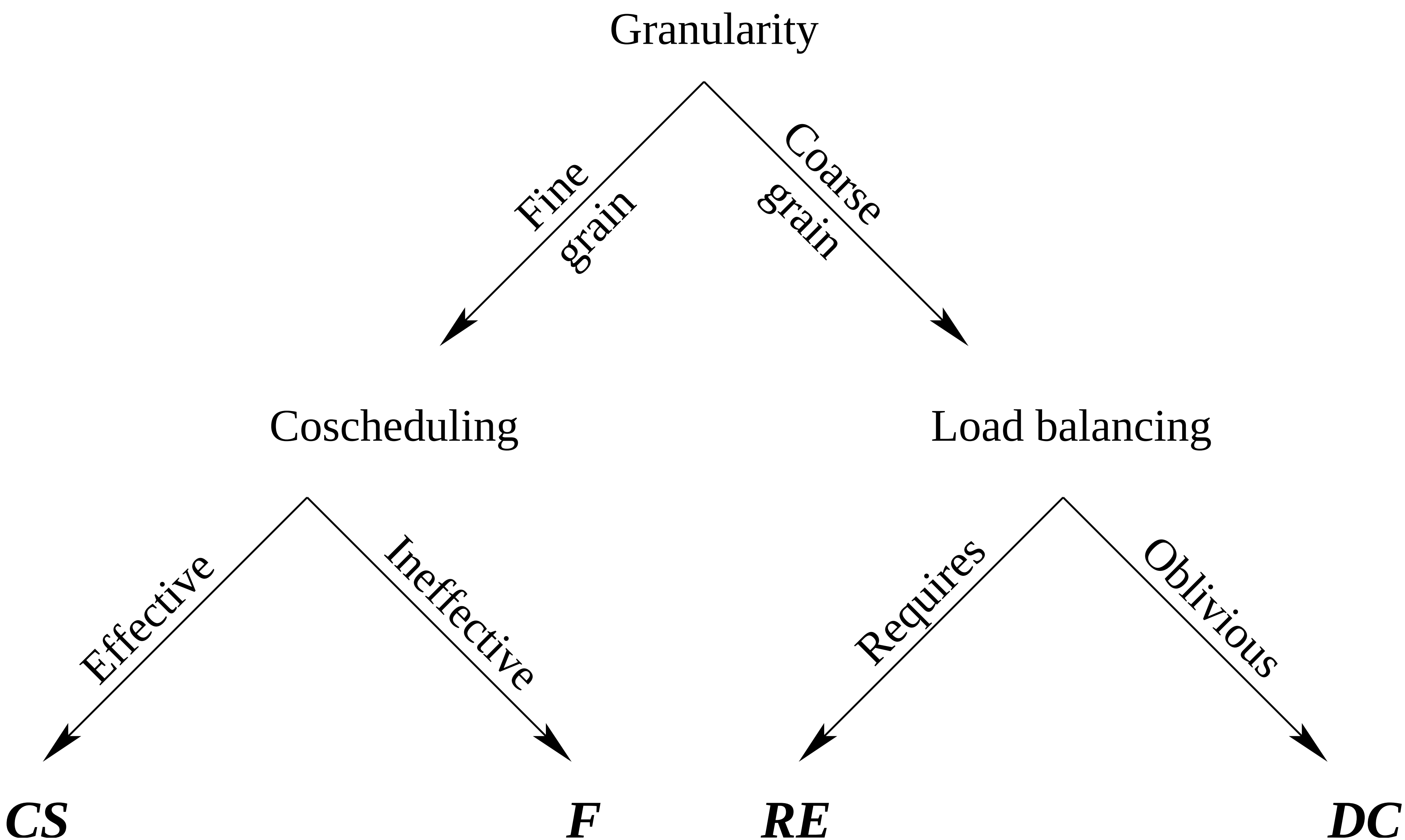
Time-Slicing Scheduling



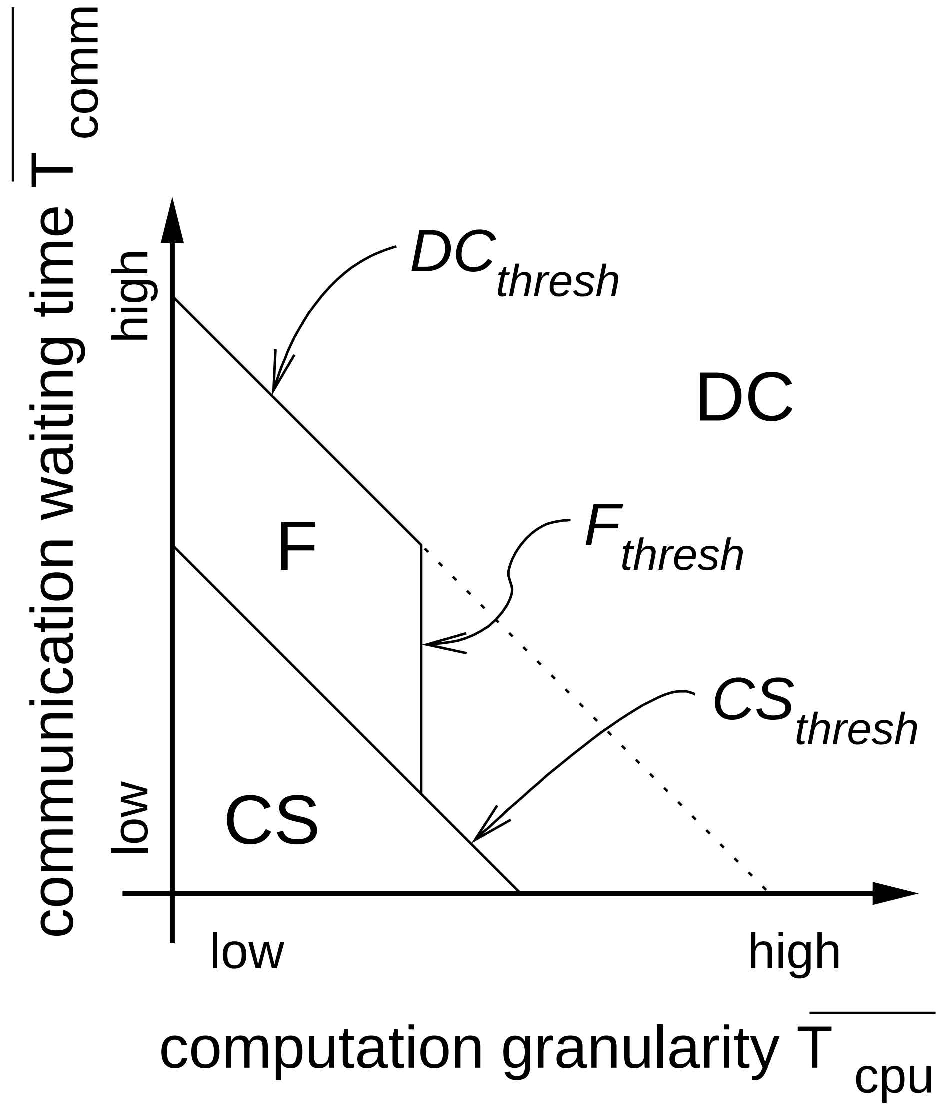
Flexible Coscheduling (FCS)

- Use global coordination with local information
- Monitor processes' communication activity
- Classify processes based on communication
- Schedule processes according to their needs

FCS Decision Tree



FCS Phase Diagram



FCS Scheduling

Use regular time-slices, but schedule processes based on classification:

- Fine-grained (CS) use explicit coscheduling
- Coarse-grained (DC) use no coordination
  - Local UNIX scheduler
- Load-imbalanced (F) use implicit coscheduling
  - Prioritized Spin-Block

Efficient Job Scheduling with STORM

FCS fully implemented with STORM - Scalable Tool for Resource Management

- Lightweight mechanisms, using HW collective communication primitives
- Extremely scalable - “local” context-switch and job launching costs on thousand of nodes
- Set of layered, modular dæmons (per node and per machine)
- “Pluggable” scheduling algorithms: Batch, Backfilling, Gang-Scheduling, Spin-block, Local, FCS, BCS

Performance Evaluation

1. Verification tests - synthetic applications based on BSP model
2. Static workloads with real applications
3. Dynamic workloads

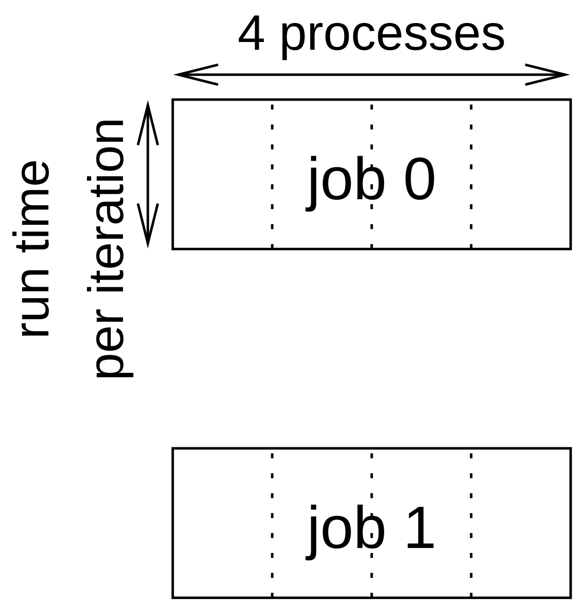
FCS compared to GS, SB, FCFS, and Local

Run on the 'Crescendo' cluster:

- 32 Dual Pentium-III 1-GHz, 1-GB RAM
- Quadrics Elan3 NICs and switch

Fine-Grained Jobs

- ▷ Two fine-grained jobs run concurrently on same nodes
- ▷ Each job computes & communicates every $5ms$ ($60s$ total)
- ▷ 2 nodes, 4 processors

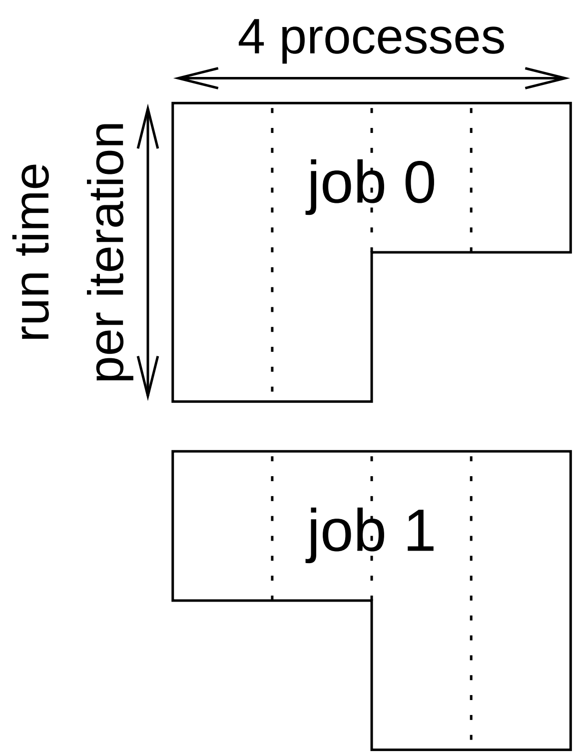


Fine-Grained Jobs - Turnaround Time

Algorithm	Job 0	Job 1	Total
FCFS	60.00	120.0	120.0
Local	234.8	231.0	234.8
GS	118.1	118.1	118.1
SB	125.4	125.4	125.4
FCS	118.3	118.4	**118.4**

Load-Imbalanced Jobs

- ▷ Same two jobs, but with load-imbalance
- ▷ Half the processes compute twice as much
- ▷ Complementing halves create opportunity for packing

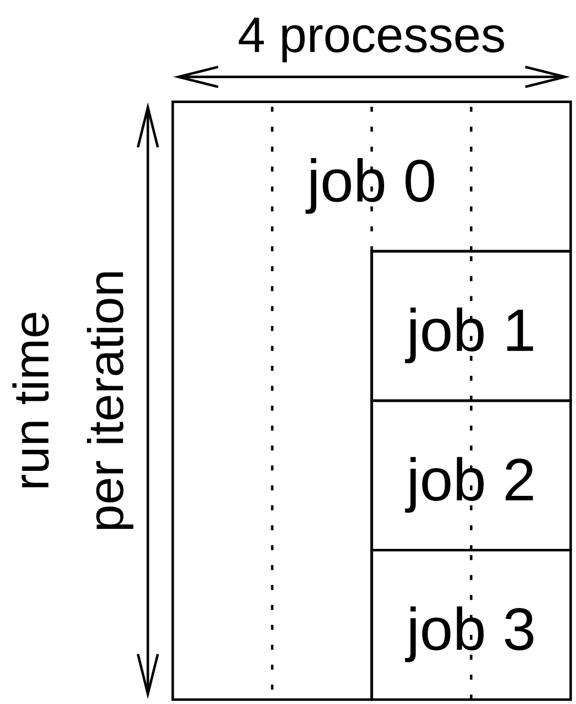


Imbalanced Jobs - Turnaround Time

Algorithm	Job 0	Job 1	Total
FCFS	116.6	233.6	233.6
Local	301.8	300.8	301.8
GS	231.3	231.9	231.9
SB	177.9	179.5	179.5
FCS	176.3	177.6	**177.6**

Complementing Jobs

- ▷ Four jobs, one with load-imbalance
- ▷ Half the processes compute four times as much
- ▷ Complementing parts create opportunity for packing



Complementing Jobs - Turnaround Time

Algorithm	Job 0	Job 1	Job 2	Job 3	Total
FCFS	231.3	290.2	349.8	408.6	408.8
Local	356.1	233.1	233.6	233.7	356.1
GS	404.7	232.1	232.2	232.2	404.7
SB	261.2	229.2	229.2	229.2	261.2
FCS	236.3	233.4	233.5	232.0	**236.3**

SWEEP3D Performance

- Particle transport code from the ASCI workload
- Balanced, fine-grained BSP application
- In this test: run time of $\approx 48s$ with $3.5ms$ granularity
- Four concurrent copies on entire cluster (64 PEs)

SWEEP3D Performance

- Particle transport code from the ASCI workload
- Balanced, fine-grained BSP application
- In this test: run time of $\approx 48s$ with $3.5ms$ granularity
- Four concurrent copies on entire cluster (64 PEs)

Algorithm	Total
FCFS	193.0
GS	194.6
SB	208.5
FCS	**197.5**

SAGE Performance

- Grid Eulerian hydro code from the ASCI workload
- Imbalanced, variable granularity
- Three concurrent copies, different input parameters
- Dedicated run times of about $39s$, $86s$, and $95s$ ($\approx 220s$ total)

SAGE Performance

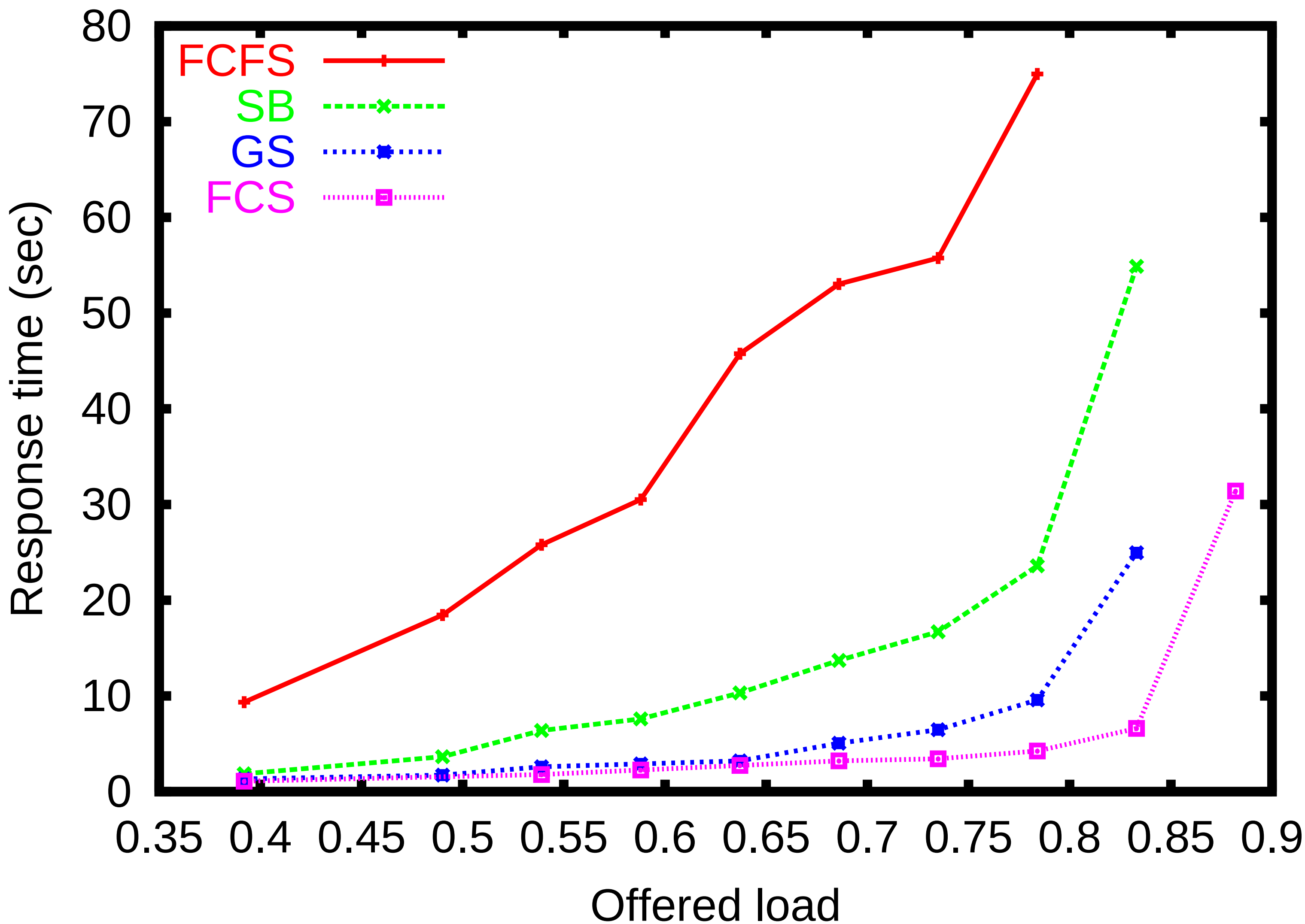
- Grid Eulerian hydro code from the ASCI workload
- Imbalanced, variable granularity
- Three concurrent copies, different input parameters
- Dedicated run times of about $39s$, $86s$, and $95s$ ($\approx 220s$ total)

Algorithm	Job 0	Job 1	Job 2	Total
FCFS	39.2	125.4	220.2	220.2
GS	120.4	222.0	227.0	227.0
SB	124.2	190.0	200.5	200.5
FCS	112.9	195.0	205.8	**205.8**

Dynamic Workload

- 1000 jobs with dynamic job arrivals, sizes and runtimes
- Based on detailed model [Lublin01]
- Synthetic test application with different granularities from $5ms$ to $500ms$
- Modify offered load by factoring run times
- Multiprogramming level of 6
- Tiimeslice of $50ms$

Dynamic Workload - Response Time

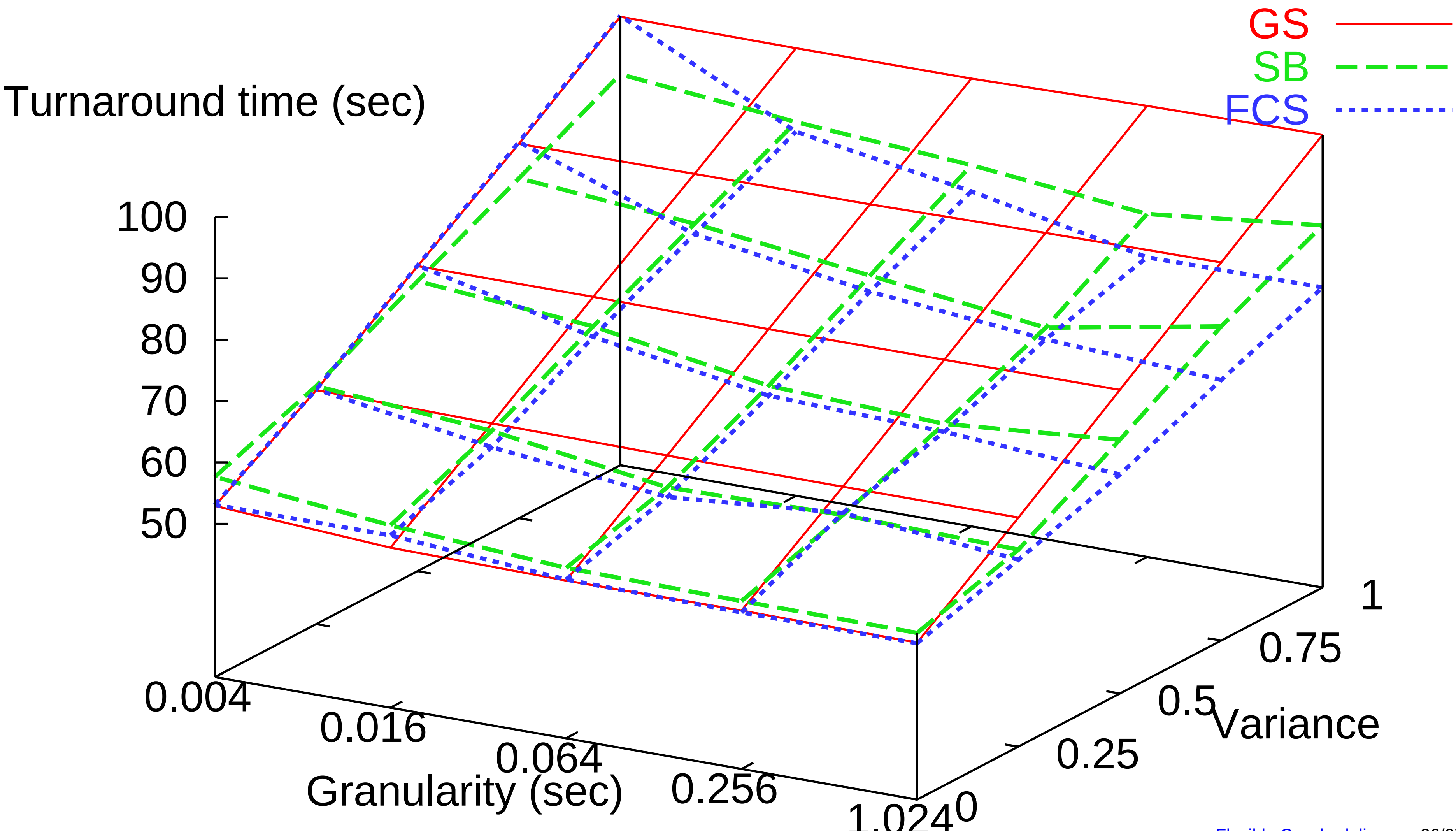


Conclusions

- FCS designed to combine the best of both worlds: explicit and implicit coscheduling.
- Monitor processes and schedule according to needs.
- Competitive with batch, local, gang, and implicit scheduling methods in varied scenarios
- Improved job packing and handling of load-imbalance lead to lower loads and better response times.

For more information:
`http://www.cs.huji.ac.il/~etcs`
email: etcs@cs.huji.ac.il

Parameter Space



STORM Demo at SC'02