



Faculty of Sciences,  
Technology  
and Communication

# Energy Conscious Scheduling

**Johnatan E. Pecero**

University of Luxembourg



**Green Days @ Lyon**



January 18, 2012



# GreenIT Project at Uni Luxembourg

- FNR CORE Project
- To provide a holistic autonomic energy-efficient solution to manage, provision, and administer the various resources within large-scale distributed systems

# GreenIT Project at Uni Luxembourg

- Aims: to develop
  - Meta-models of Cloud Computing (CC) systems
  - Resource management methodologies in CC
    - Scheduling, resource allocation, load balancing
  - Autonomic resource management for CC

# Plan

- Context and Motivation
- Energy Conscious Scheduling
- Energy-aware Scheduling Algorithms
- Concluding Remarks



# Background

Top500.org data  
(November 2011)



- K supercomputer SPARC64 (No. 1) power : 12.65 MW
- Average top 10 supercomputer power: 4.6 MW
- More than 5% of top500 supercomputers consume more than 1 MW

# 2012 Energy Consumption in Data Centers



DatacenterDynamics.com

- Data centers **currently consume** about **31 GW**
  - The average total power to rack is about 4.05 kW
  - Around 58% of racks consuming 5kW per rack
  - 28% of racks consuming from 5kW to 10kW
  - 14% of racks consuming more than 10kW per rack
- A **projected** rate of **increase** in energy consumption of **19%** into 2012 for world's data center
    - More than 10% (300 MW) in France (2011-2012)



100 Units  
Source  
Energy

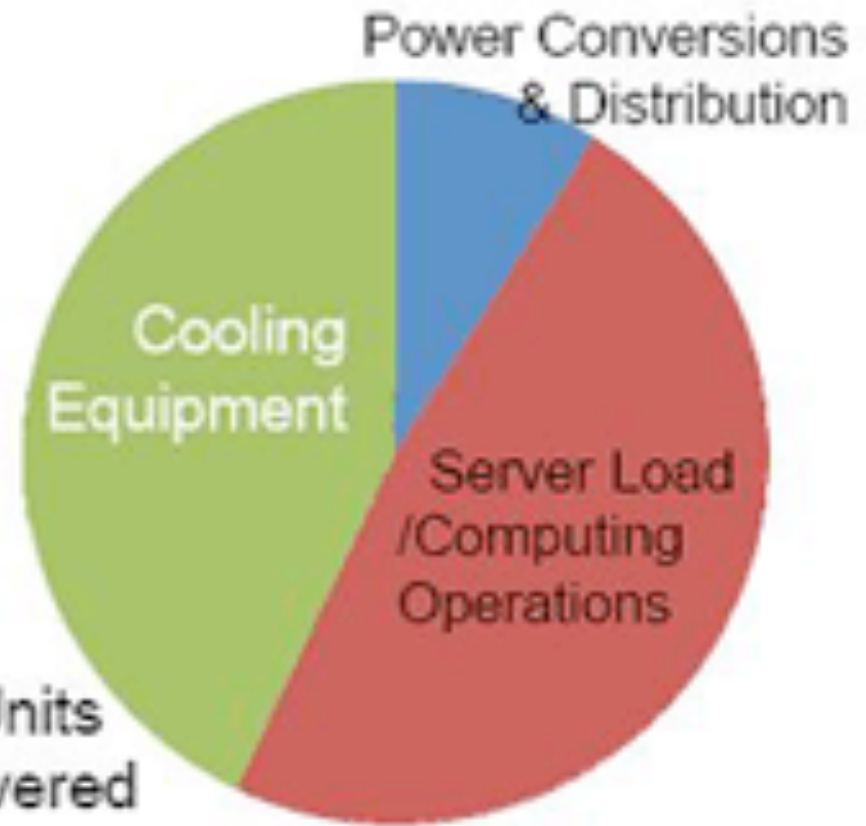
# Typical Data Center Energy End Use



35 Units  
Power Generation

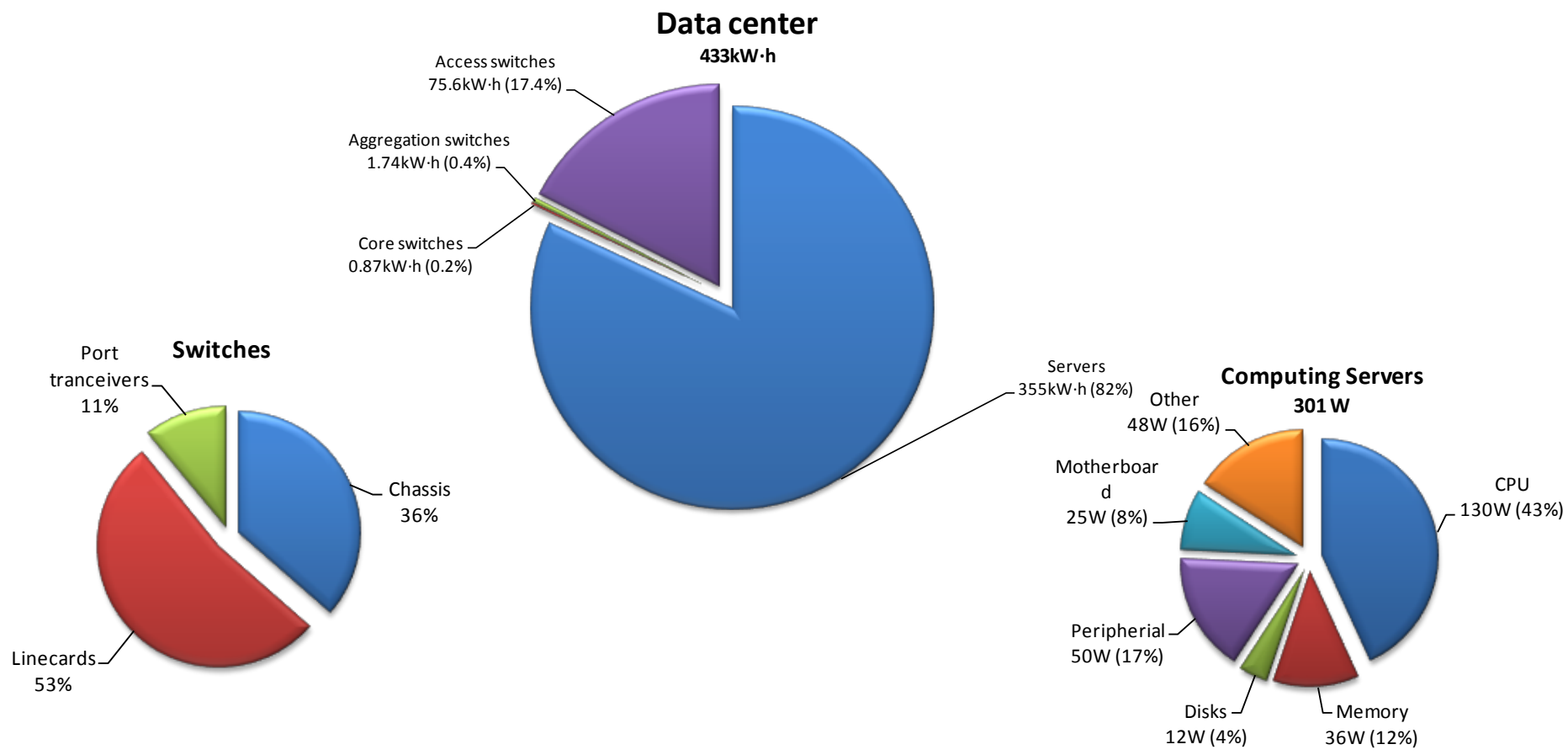


33 Units  
Delivered



- Source : U.S. Federal Energy Management Program
- ([http://www1.eere.energy.gov/femp/program/dc\\_energy\\_consumption.html](http://www1.eere.energy.gov/femp/program/dc_energy_consumption.html))

# ■ Distribution of energy consumption in data center



Source: GreenCloud Simulator, University of Luxembourg

# Green Approaches

- **Hardware approach**

- Energy-efficient Microprocessors, multi-core
- Better CPU Power Management, Power Heterogeneous Processors
- Solid State Disks
- Energy-efficient Monitors

- **Software approach**

- Virtualization
- Energy Conscious Scheduling and Resource Allocation (S&RA)
- Energy-aware Algorithm Design

- **Cloud Computing**

- Redesigning data centers, (e.g. google, microsoft)

# Green Approaches

- **Hardware approach**

- Energy-efficient Microprocessors, multi-core
- Better CPU Power Management, Power Heterogeneous Processors
- Solid State Disks
- Energy-efficient Monitors

- **Software approach**

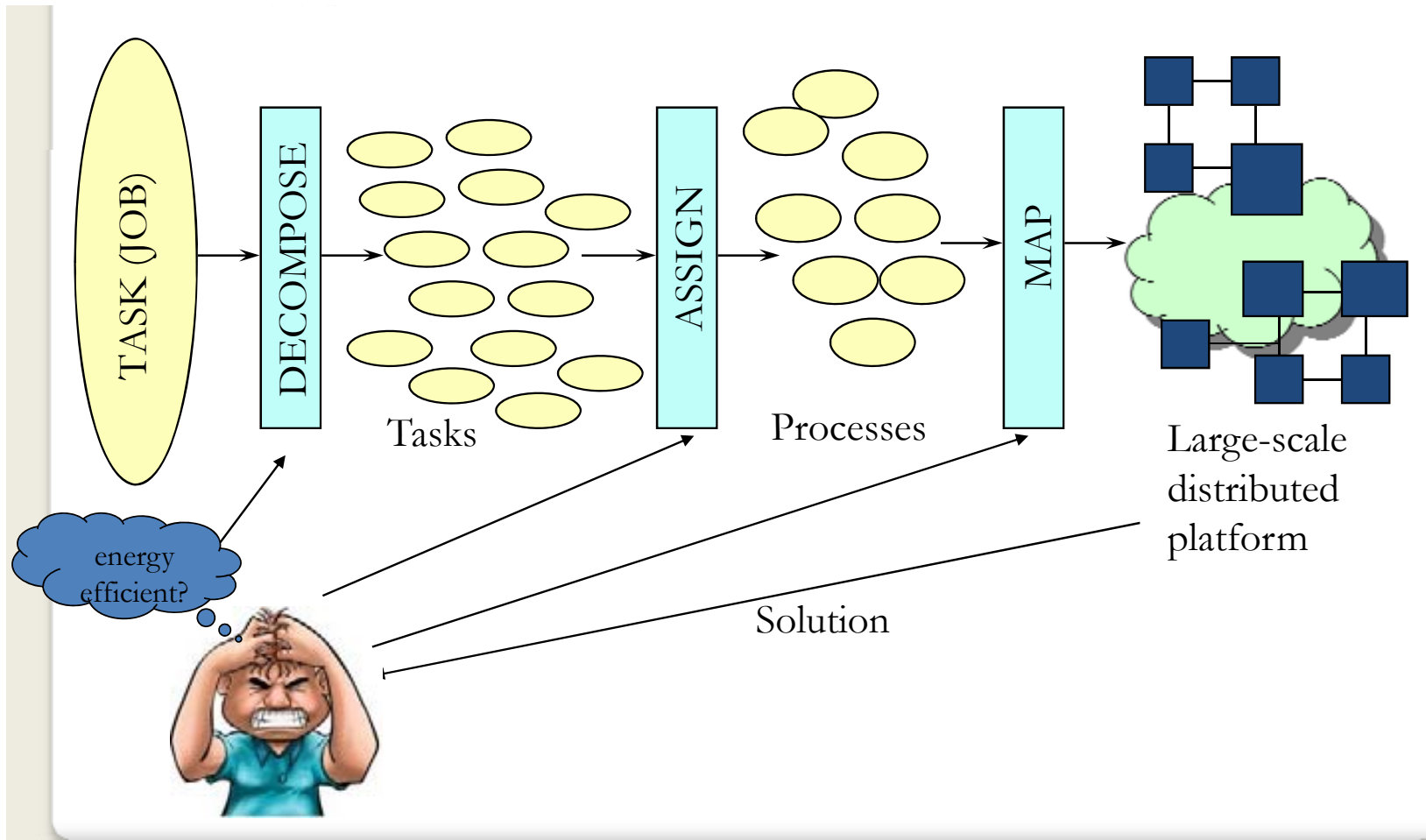
- Virtualization
- Energy Conscious Scheduling and Resource Allocation (S&RA)
- Energy-aware Algorithm Design

- **Cloud Computing**

- Redesigning data centers, (e.g. google, microsoft)



# Energy Conscious Scheduling and Resource Allocation for Large-Scale Distributed Systems



Source: © Albert Y. Zomaya's talk, University of Luxembourg

# Energy Conscious Scheduling



# Scheduling Basis

- Scheduling deals with the allocation of scarce resources to tasks over time, subject to a set of constraints
- The main constraints are resource constraints and precedence constraints between activities

# Traditional Scheduling - Machine environment

- Single machine and machines in parallel
  - Single machine (uniprocessor systems)
  - $P_m$  identical parallel machines (homogeneous)
  - $Q_m$  machines in parallel with different speeds
  - $R_m$  unrelated machines in parallel (heterogeneous)

# Traditional Scheduling - Objectives

- Performance measures of individual jobs
  - $C_j$  Completion time of a job  $j$
  - $L_j$  Lateness
  - $T_j$  Tardiness
  - $E_j$  earliness
  - $U_j$  unit penalty = 1 if the completion time of a job is greater than due date

# Traditional Scheduling - Objectives

- Functions to be minimized (QoS related)
  - $C_{\max} = \max C_j$  makespan
    - The total amount of time required to complete a group of jobs
  - $L_{\max} = \max L_j$  maximum lateness
  - $\sum w_j T_j$  = Total weighted tardiness (Past due date)
  - $\sum C_j$  Flow time (user metric)
    - throughput time, or time spent in the system
  - $\sum w_j C_j$  Total weighted completion time

# Scheduling Problem

## Scheduling Problem.

Determine  $\sigma$  : when and where the computational units (tasks) will be executed.

## Theorem

*Minimizing the makespan (basic problem) is NP-Hard [Ullman75]*

- Solutions may be obtained by exact methods, heuristic, meta-heuristic method, approximated methods

# Traditional Scheduling - LS Algorithm

---

## Algorithm 2.3 List Scheduling

---

Calculate the priority of each task  $t_i \in V$  according to some predefined scheme.

Sort tasks  $t_i$  into list  $L = \{t_1, t_2, \dots, t_n\}$  by decreasing order of their priorities and precedence constraints.

**While**  $L$  is not empty

    Remove the first task from  $L$  and assign it to an appropriate processor in order to optimize a predefined cost function.

**return** (*schedule*)

---

Task Selection  
Phase

Processor  
Selection Phase

- Theorem : The List Scheduling algorithm is a 2-approximation for MAKESPAN Scheduling on identical machines
- Some List Scheduling Priorities:
  - Critical Path Method, Longest Path, Longest Processing Time

# Energy-aware Scheduling at Site Level

# Scheduling with Energy considerations

- Three optimization problems
  - P1: Optimize performance (QoS related objective) subject to an Energy Budget
  - P2: Optimize energy without performance deterioration
  - P3: Optimize performance and energy simultaneously



# Energy-aware Scheduling

- As computation time cost, better scheduling is intrinsically more **green**
  - maximizing resource utilization, avoiding idle time, task and resource consolidation
- Popular energy saving techniques combined with scheduling
  - Dynamic power management
    - When a device is idle, it can transition to low-power sleep states..., switch off/on
  - **Dynamic Voltage Scaling**
    - A device can be run at different speeds with different usage rates
    - Execution of jobs can be slowed down to save power as long as all jobs are completed by their deadline

# Dynamic Voltage Scaling



- DVS enables processors to dynamically changing its working voltage and frequency without stopping or pausing the execution of any instruction.
  - During some time slots
    - Idle time
    - Communication phases

- Modern components allow voltage regulation
  - Bios
  - Application such as PowerStrip
- The aim of DVS is to reduce energy consumption
- This reduction is achieved at the expense of sacrificing clock frequencies; therefore longer time is will be required to execute a given application

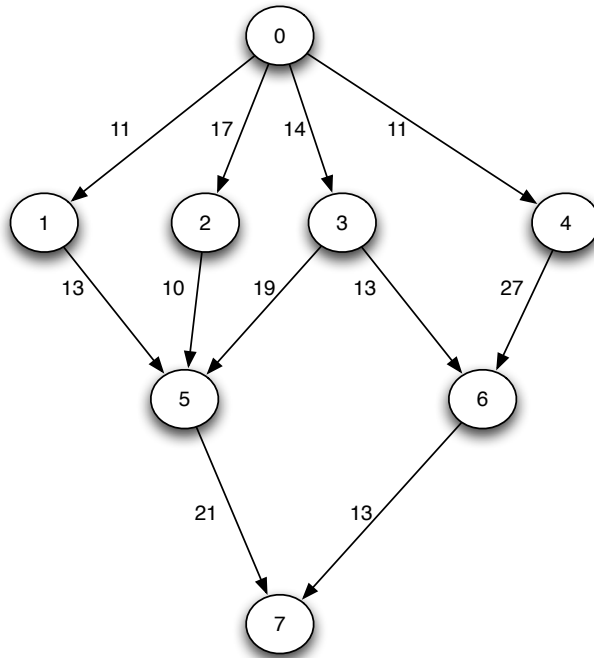
# System Model

- A set  $M$  of  $m$  heterogeneous and DVS-enabled processors that are fully interconnected

**Table:** Voltage-Relative Speed Pairs

Level	Pair 1		Pair 2		Pair 3		Pair 4		Pair 5		Pair 6	
	Volt. ( $v_k$ )	Rel. Speed (%)	Volt. ( $v_k$ )	Rel. Speed (%)	Volt. ( $v_k$ )	Rel. Speed (%)	Volt. ( $v_k$ )	Rel. Speed (%)	Volt. ( $v_k$ )	Rel. Speed (%)	Volt. ( $v_k$ )	Rel. Speed (%)
0	1.50	100	2.20	100	1.50	100	1.75	100	1.20	100	1.35	100
1	1.20	80	1.90	85	1.40	90	1.40	80	1.15	90	1.25	85.7
2	0.90	50	1.60	65	1.30	80	1.20	60	1.10	80	1.20	71.5
3			1.30	50	1.20	70	0.90	40	1.05	70	1.10	57.1
4			1.00	35	1.10	60			1.00	60	0.9	32.2
5					1.00	50			0.90	50		
6					0.90	40						

# Application Model



task	$m_0$	$m_1$	$m_2$	$\bar{p}_i$	$b\text{-level}$
0	11	13	9	11	101
1	10	15	11	12	67
2	9	12	14	11.67	63.67
3	12	16	10	12.67	73.67
4	15	11	19	15	79
5	13	9	5	9	42
6	11	15	13	13	37
7	11	15	10	12	12

**Figure:** In the left a sample Directed Acyclic Graph. In the right a table with base execution time of the tasks at maximum voltage.

## Communication to Computation Ratio (CCR)

# Energy Model

- Derived from the power consumption model in complementary metal-oxide semiconductor (CMOS) logic circuit
- Capacitive Power  $P_c = AC_{ef}v^2 f$ , (5)

A: #switches per clock cycle, C: total capacitive load  
V: supply voltage, f: frequency

- Our Energy Model

$$E_c = \sum_{i=1}^n AC_{ef}v_i^2 f \cdot p_i^* = \sum_{i=1}^n \gamma v_i^2 p_i^*, \quad (6)$$

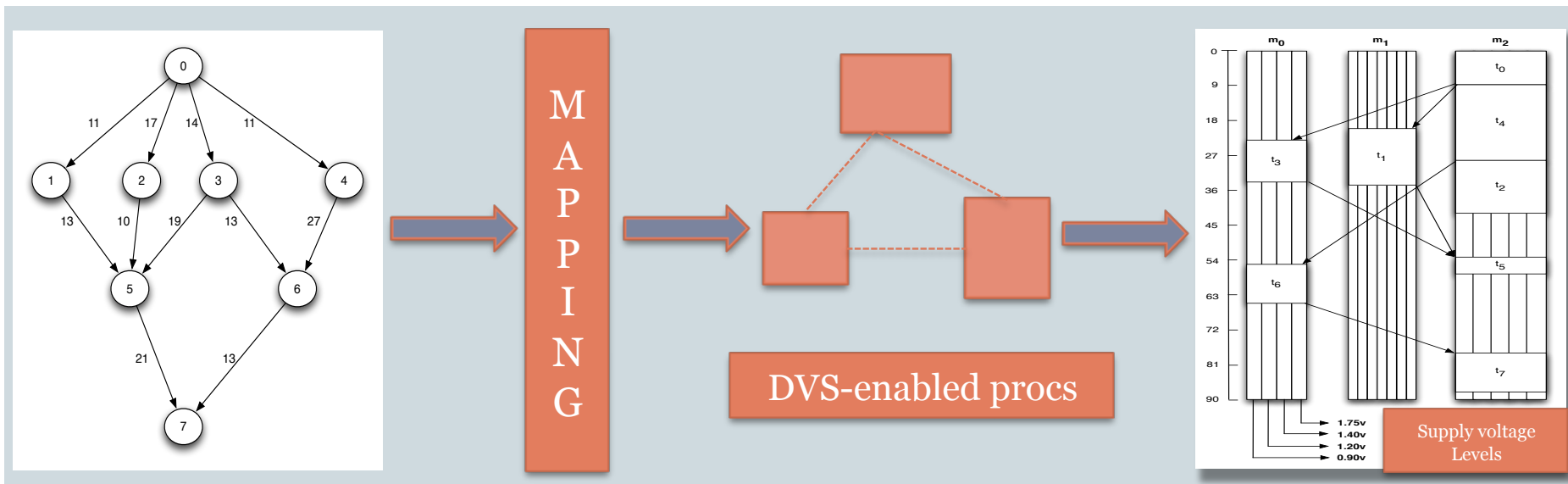
$$E_i = \sum_{j=1}^m \sum_{idle_{jk} \in IDLE_j} \gamma v_{j,low}^2 I_{jk}, \quad (7)$$

- Total Energy

$$E_t = E_c + E_i. \quad (8)$$

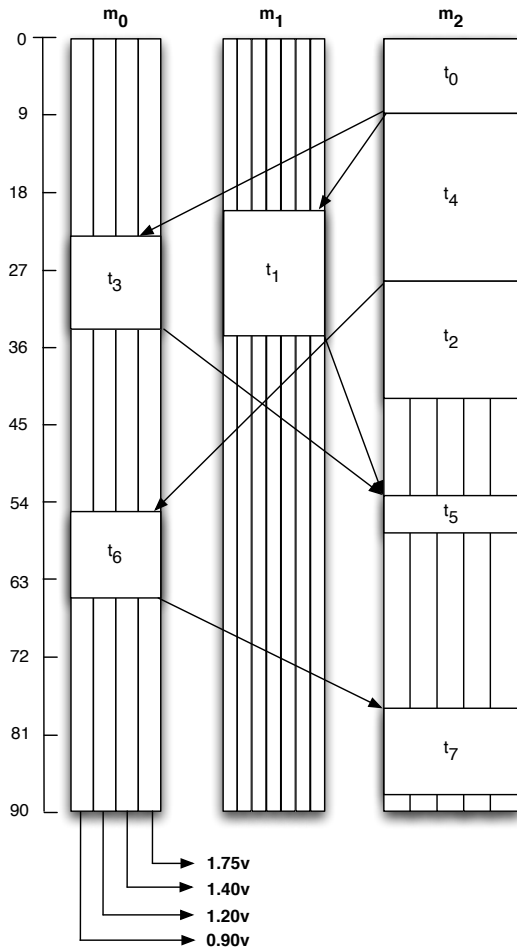
# Scheduling Model

- Allocation of a set  $N$  of  $n$  tasks to a set  $P$  of  $p$  processors (without violating precedence constraints) aiming to minimize schedule length (i.e. Earliest Finish Time - EFT) and energy consumption



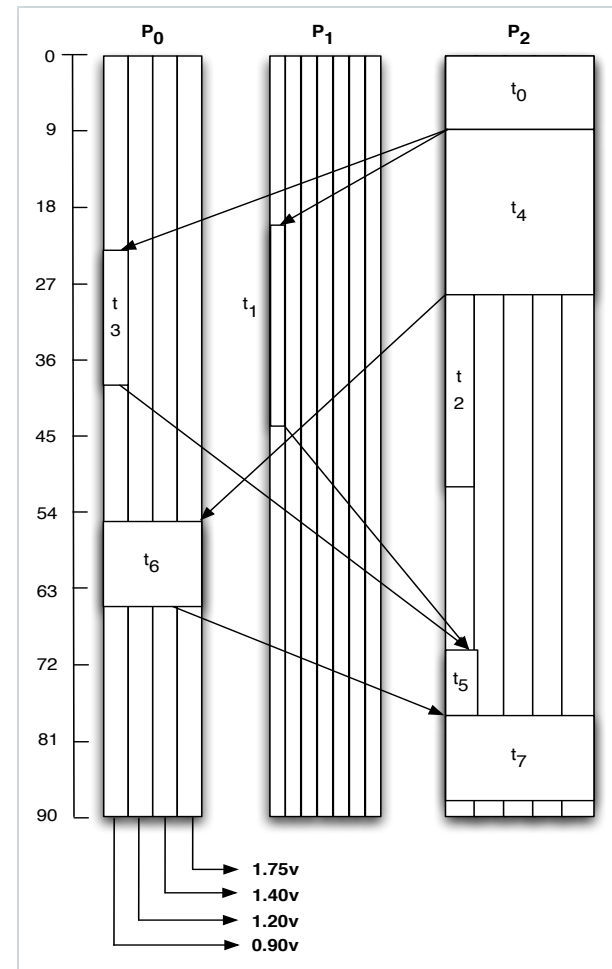
# Energy-aware Scheduling (Best-effort + Slack Reclamation)

## Best-effort



Makespan = 89,  
Energy = 380

## With DVS



Makespan = 89,  
Energy = 333

# Problem Summary

- Scheduling: performance EFT
- DVS: energy optimization
- Contradictory Objectives:
  - Speed of computation vs energy consumption
- Multi-objective approach is a necessity
- The aim
  - To provide Decision Maker (DM) a set of possible schedules to choose from
- DM can offer a set of price based services by Service Level Agreement
  - Price - energy
  - Quality of Service (QoS) - EFT



# Pareto Dominance

- Given a set of  $M$  objectives  $f_1, f_2, \dots, f_M$  to be minimized, solution  $s_1$  weakly dominates solution  $s_2$ , denoted  $s_1 \preceq s_2$ , whenever :
  - The solution  $s_1$  is no worse than  $s_2$  in all objectives or  $f_i(s_1) \leq f_i(s_2) \ i \ \{1, 2, \dots, M\}$ .
  - If, in addition, there exists a  $j \ \{1, 2, \dots, M\}$  such that  $f_j(s_1) < f_j(s_2)$ , then  $s_1$  strictly dominates  $s_2$ , denoted  $s_1 \prec s_2$ .

# Proposed Solution

- A solution founded on GRASP framework
  - Multi-objective
    - (CGC 2011, Sydney, Australia)
- Two phases search procedure
  - First phase principle
    - Best-effort idea
  - Second phase
    - The schedules in the first phase are scrutinized using DVS by a local search

# Algorithm 1. Multi-objective GRASP.

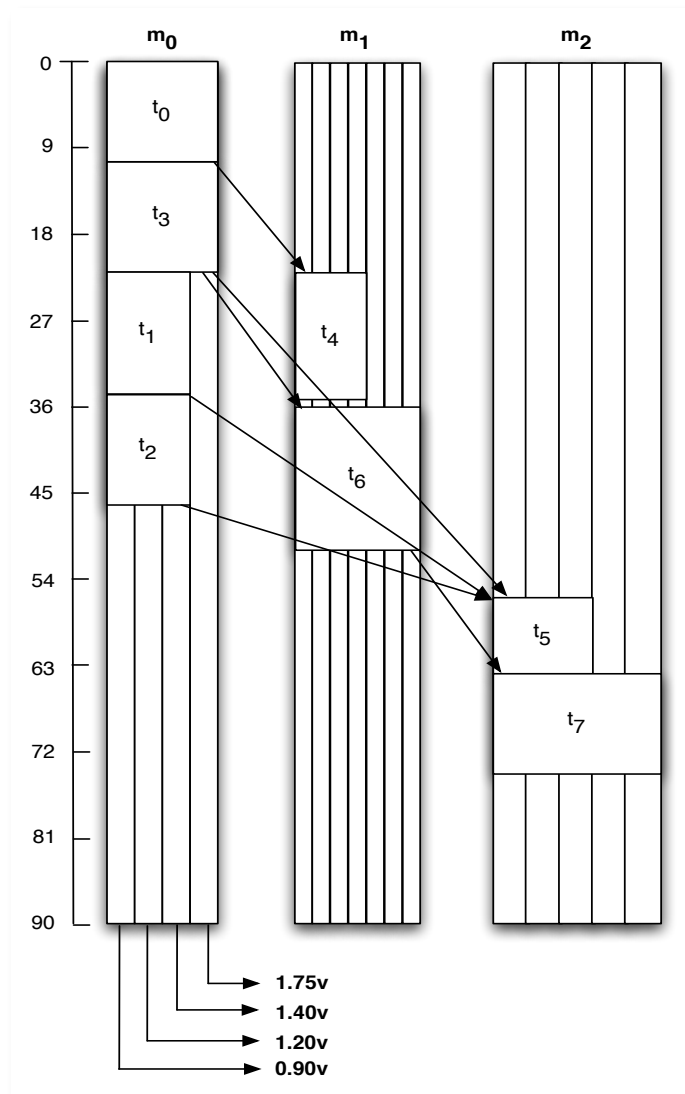
```
1: function MOGRASP( $G = (T, E), N, I$ )
2:    $bestFront := \emptyset$ 
3:    $L := \mathbf{ComputeListPriority}(G = (T, E))$ 
4:   for  $x:=1$  to  $N$  do
5:      $newFront := \mathbf{OneGeneration}(G = (T, E), L, I)$ 
6:      $bestFront := bestFront \cup newFront$ 
7:     Remove dominated solutions from  $bestFront$ 
8:   end for
9:   return  $bestFront$ 
10: end function
11: function OneGeneration( $G = (T, E), L, I$ )
12:    $solution := \mathbf{ConstructSolution}(G = (T, E), L)$ 
13:    $solution' := \mathbf{VoltageScaling}(solution)$ 
14:    $front := \mathbf{MOLocalSearch}(solution', I)$ 
15:   return  $front$ 
16: end function
```

Computational Complexity  $O(|e| + |n| + |n|lg|n| + N(|m|(e + n) + |m||TS| + |I||MAXSTEPS|(e + n) + |K||ND|))$ .

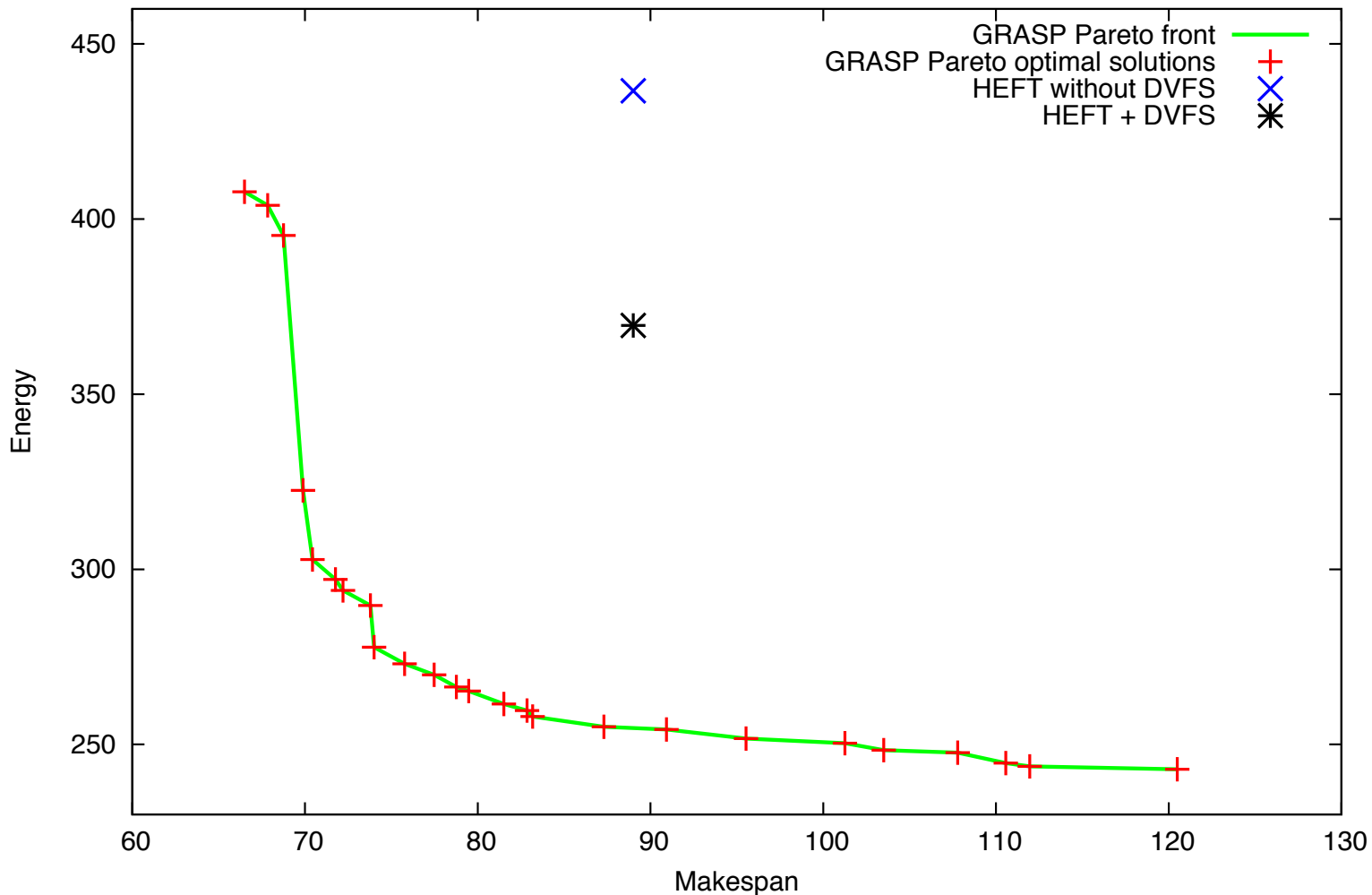
**Algorithm 4.** Multi-objective Local Search Function.

```
1: function MOLocalSearch(solution, I)
2:   front := solution
3:   for iter :=1 to I do
4:     Select a task  $t_i$  at random from solution
5:     for searchstep :=1 to MAXSTEPS do
6:       ▷ Next step explores possible makespan improvement
7:       Select a processor  $p_k \neq p_j$  at random ▷  $p_j$ 
       is the current location of task  $t_i$ 
8:       Select  $(v_k, rs_k)$  from the corresponding set
       of voltage and relative speed of  $p_k$  randomly ▷ DVFS
       technique to explore energy improvement
9:       Allocate  $t_i$  on  $p_k$  with voltage  $v_k$  and relative
       speed  $rs_k$ 
10:      solution' := Compute current EFT and En-
       ergy
11:      if solution' is not dominated by any member
       of front then
12:        front := front  $\cup$  solution'
13:        solution := solution'
14:      end if
15:    end for
16:  end for
  return front
17: end function
```

Makespan = 74,  
Energy = 236



# A sample Pareto Front Computed by GRASP



# Experimental Results

- We compare the performance of the algorithm against a best effort scheduling algorithm (Heterogeneous Earliest Task First - HEFT) and HEFT with a Dynamic Voltage Scaling technique

# Heterogeneous Earliest Finish Time (HEFT)

- List based scheduling algorithm
  - Maintain a list of all tasks according to a priority (b-level)
- Two phases:
  - First phase: a ready task is selected from the priority list
  - Second phase: A suitable processor that minimizes EFT for the task is selected
- Highly competitive



# HEFT + DVS

- Best effort idea first
  - Apply HEFT
- Then reduce voltage without increasing schedule length

# Experimental Setting

- GRASP iteration number :  $N = 200$
- Maximum number of iterations in the local search:  $l = 60$   
 $\alpha \in [0, 0.2]$  Interval close to 0 based on the best effort idea
- The parameters fixed by experiments

# Comparison Methodology

- A first solution is computed with HEFT and HEFT+DVS
- A second resolution is done with the GRASP approach to generate the Pareto solution set
- Only one solution is selected from the Optimal Pareto Set
- The solution is closest to the solution computed by HEFT and HEFT+DVS in the sense of Euclidean distance
- Final a comparison is done between the closest solution, HEFT, and HEFT+DVS
  - the gain over these solution

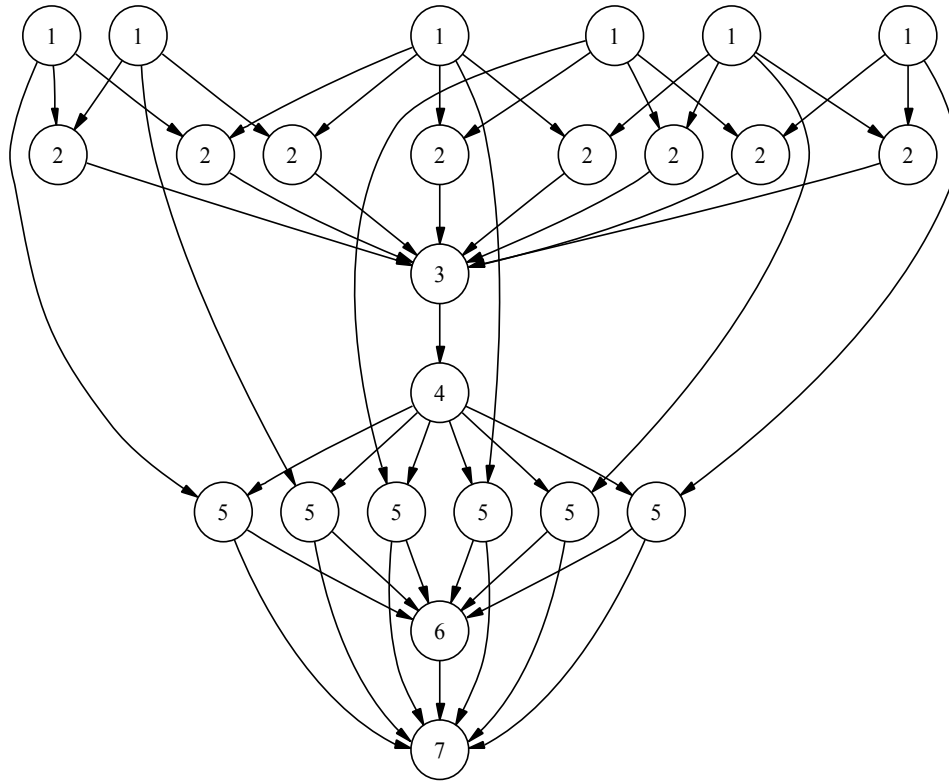
$$\text{gain} = (\text{HEFTbasedSolut} - \text{GRASPSol}) / \text{HEFTbasedSols}$$

# Simulations

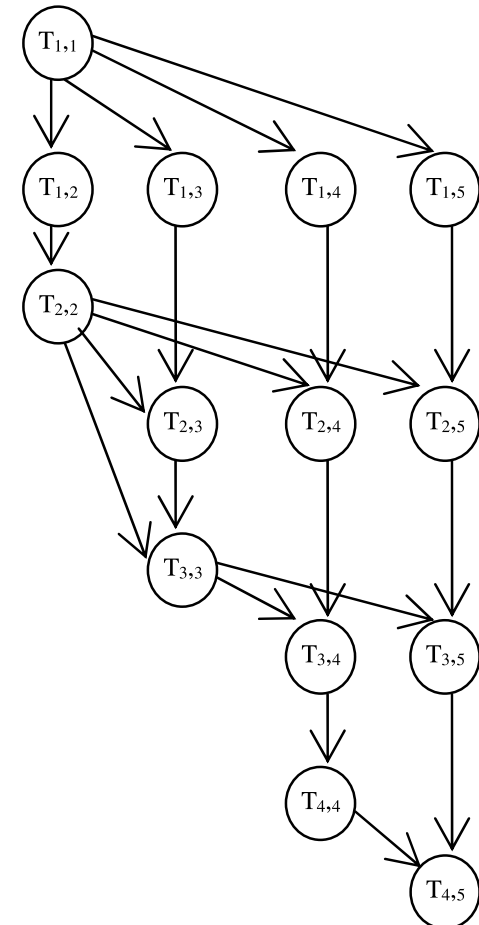
- Instances
  - Four real-world parallel applications
    - Laser Interferometer Wave Observatory (LIGO)
    - Robot
    - Sparse matrix
    - Gaussian Elimination
      - Generated synthetically
- Number of Processors (8, 16, 32)
- Five different CCRs (0.1, 0.5, 1, 5, 10)

From the Standard Task Graph Set

# Sample Instances



LIGO



Gaussian Elimination

# Employed Instances and Their Characteristics

Application	Number of Tasks	Number of Edges	ETR
LIGO	76	132	1.73
Robot Control	88	131	1.48
Sparse Matrix	96	128	0.69
GE	42	68	1.61
	52	86	1.65
	63	106	1.68
	75	128	1.70
	88	152	1.72

# Results

**Table 2:** Gain according to the number of processors

Number of Processors	Gain over HEFT		Gain over HEFT +DVS	
	Makespan(%)	Energy(%)	Makespan (%)	Energy (%)
8	7.35	12.77	7.22	8.15
16	6.95	13.36	6.86	8.61
32	8.59	14.47	8.57	11.15

# Results

**Table 3:** Gain in Real Applications

Application	Gain over HEFT		Gain over HEFT +DVS	
	Makespan(%)	Energy(%)	Makespan (%)	Energy (%)
LIGO	8.13	15.56	8.13	13.03
ROBOT	8.12	17.78	8.07	9.94
SPARSE	5.24	16.35	5.24	15.20



# Results

**Table 4:** Gain according to the CCRs

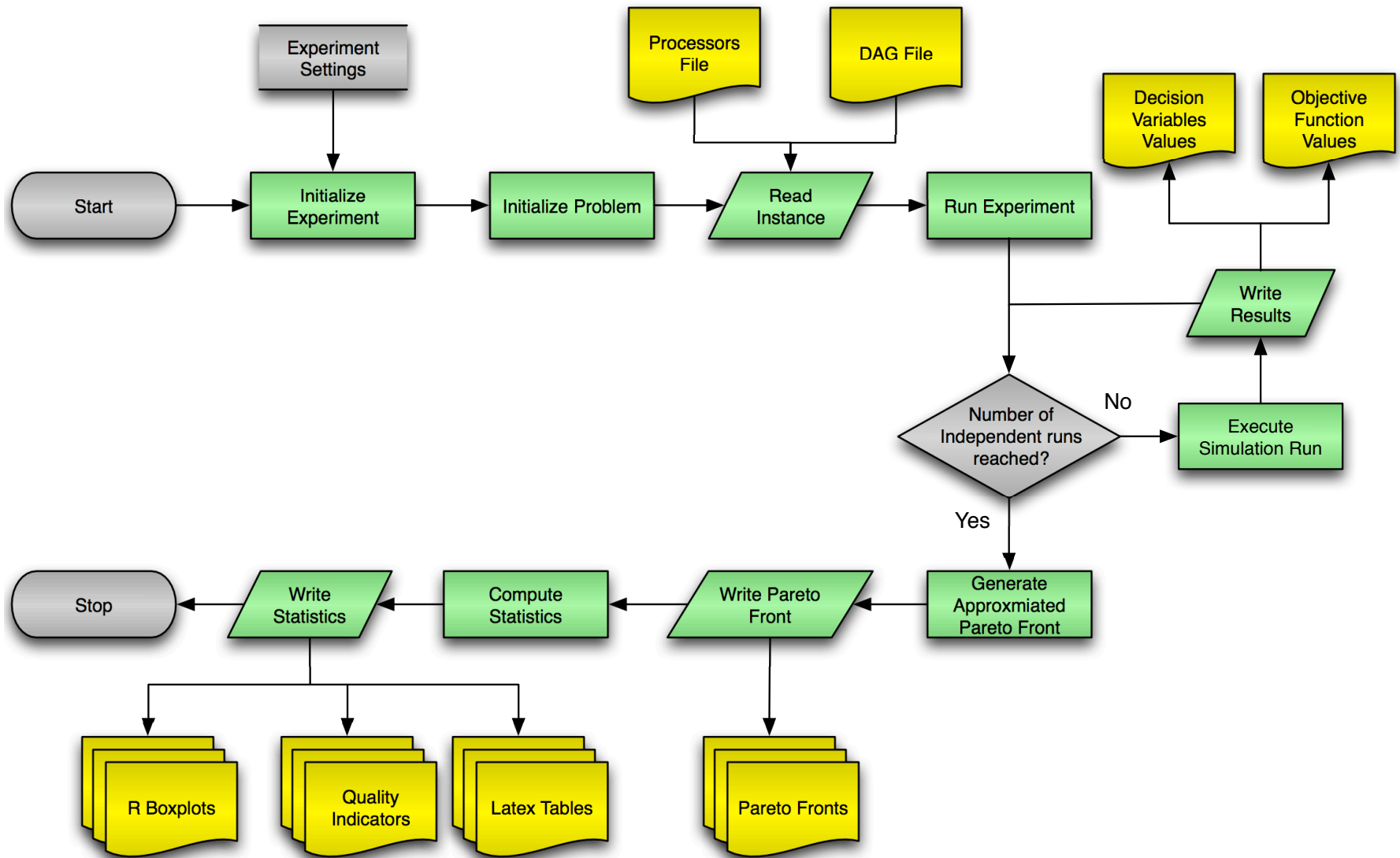
CCR	Gain over HEFT		Gain over HEFT +DVS	
	Makespan(%)	Energy(%)	Makespan (%)	Energy (%)
0.1	1.81	7.8	1.74	3.41
0.5	4.43	10.05	4.30	5.10
1	6.25	12.15	6.10	7.35
5	11.58	18.54	11.55	14.29
10	14.07	19.15	14.06	16.37

# Results

**Table 5:** Gain according to size for Gaussian Elimination Applications

Number of Tasks	Gain over HEFT		Gain over HEFT +DVS	
	Makespan(%)	Energy(%)	Makespan (%)	Energy (%)
42	8.37	12.9	8.26	8.90
52	7.35	10.89	7.18	6.74
63	7.97	11.93	7.89	7.33
75	7.59	11.64	7.46	6.72
88	8.33	11.25	8.19	6.56

# GreenMetal



# Scheduling in Grids

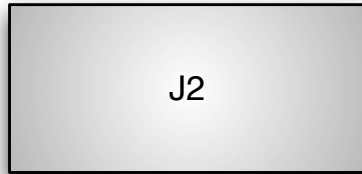
- Efficient scheduling across nodes is necessary to maximize application performance regardless of the efficiency of your parallel algorithms
- Dynamic scheduling in a heterogeneous environment is significantly more complicated
- Many unpredictable events can occur :
  - Robust schedules

# Scheduling in Grids (cont...)

- Scheduling is a key part of the workload management software which usually perform some or all of:
  - Queuing
  - Monitoring
  - Resource Management
  - Accounting
  - Scheduling

# Exploiting Heterogeneity

- Schedulers can take advantage of heterogeneity to schedule tasks efficiently and in a green mode



**Execution Time**

**P1**

**P2**

**2**

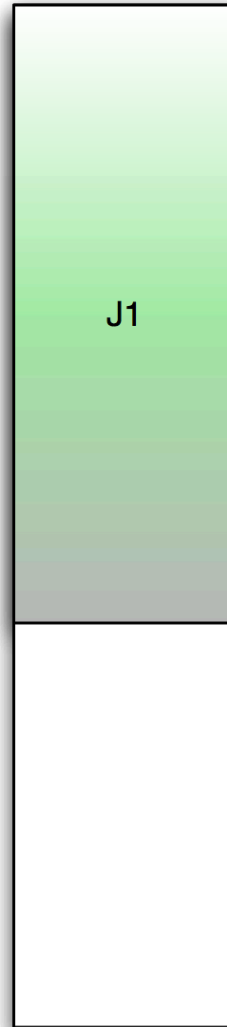
**5**

**Power Consumption**

**P1 Emax(W) = 129    P2 Emax(W) = 109**

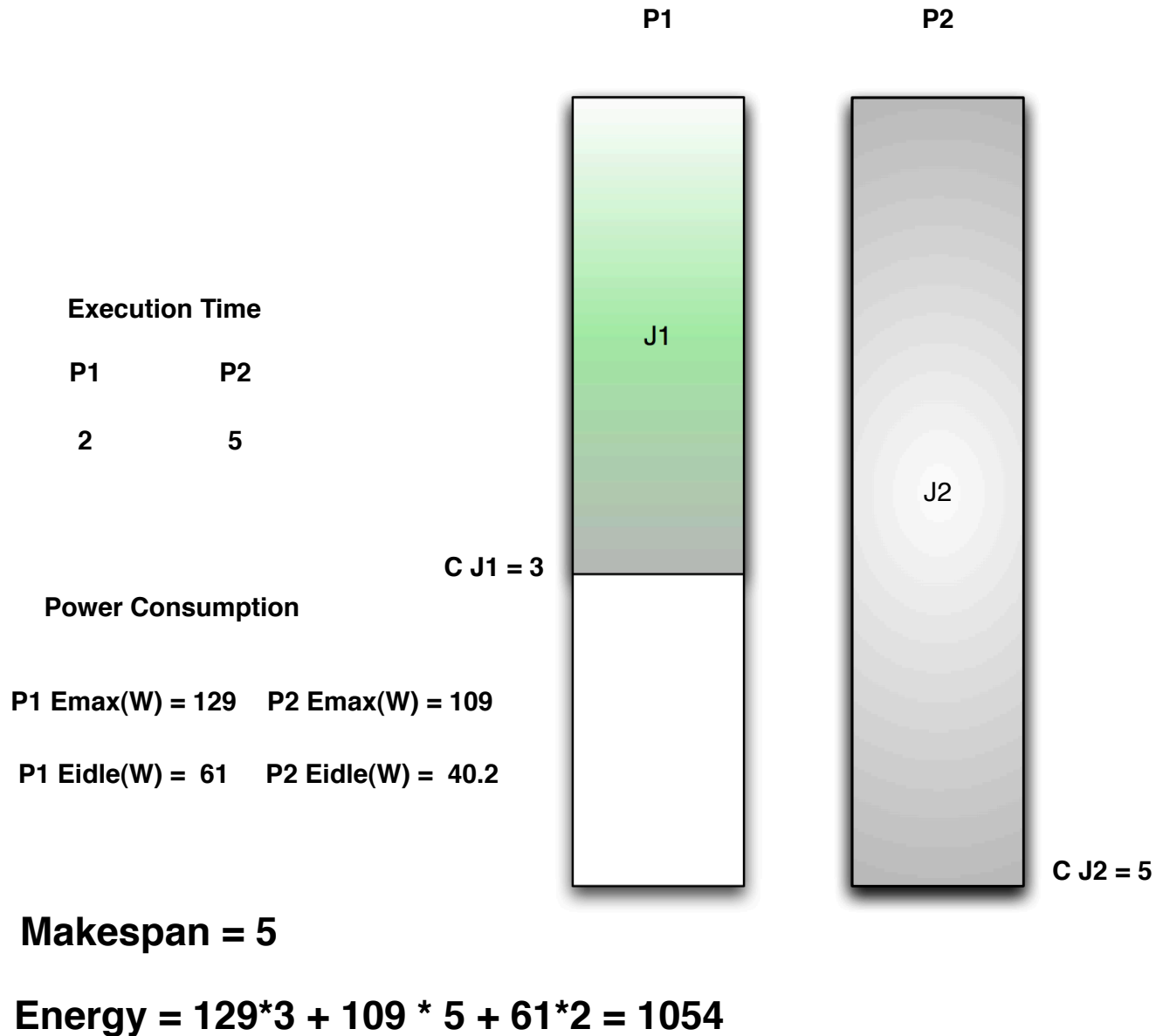
**P1 Eidle(W) = 61    P2 Eidle(W) = 40.2**

**C J1 = 3**



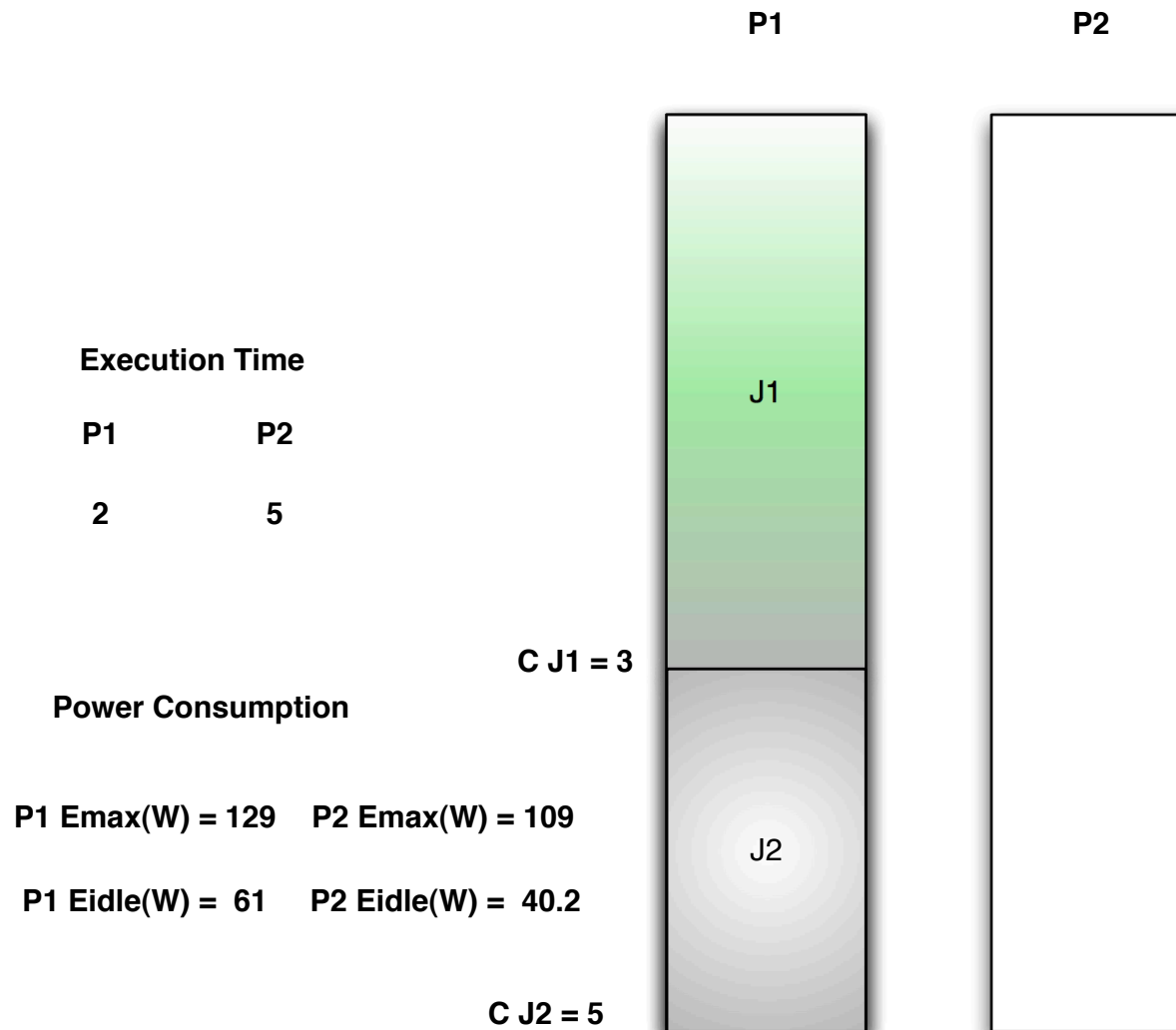
$$\sum_{\substack{t_i \in T: \\ f(t_i) = m_j}} EC(t_i, m_j) + \sum_{m_j \in P} EC_{IDLE}(m_j)$$

# First Scenario





# Second Scenario



# Low complexity heuristics

---

**Algorithm 1** Pseudo-code for the low-cost heuristics

---

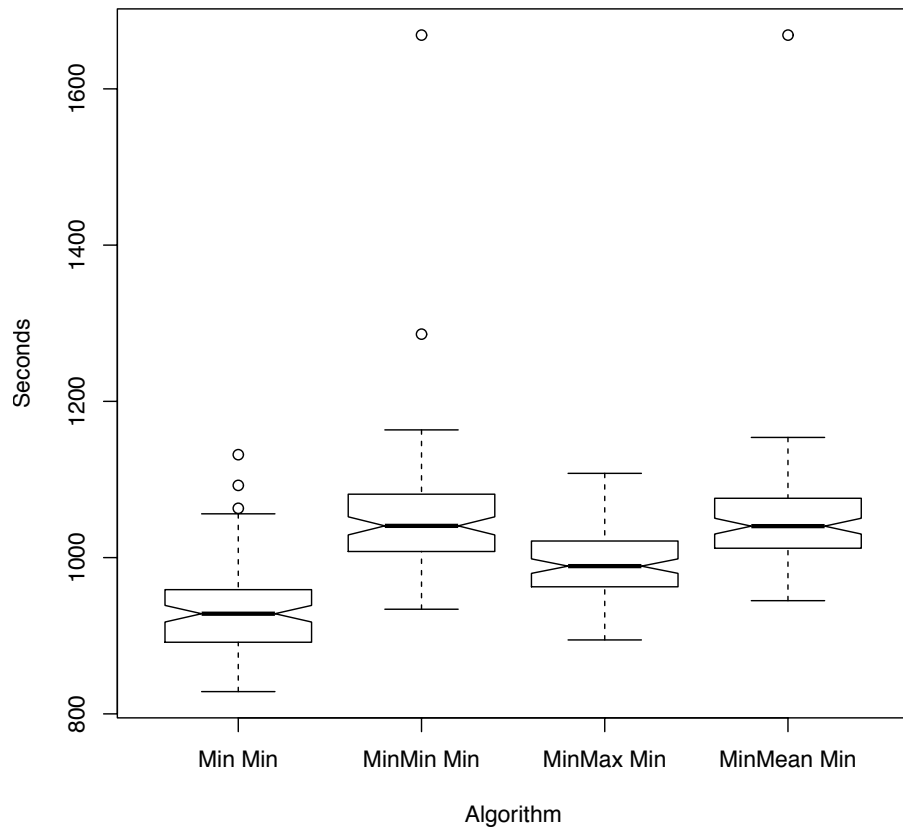
- 1: Compute **Priority** of each task  $t_i \in T$  according to some predefined objective;
  - 2: Build the list  $L$  of the tasks sorted in decreasing order of Priority;
  - 3: **while**  $L \neq \emptyset$  **do**
  - 4:   Remove the first task  $t_i$  from  $L$ ;
  - 5:   **for** each machine  $m_j$  **do**
  - 6:     Evaluate Score Function  $SF(t_i)$ ;
  - 7:   **end for**
  - 8:   Assign  $t_i$  to the machine  $m_j$  that optimize the Score Function;
  - 9:   Update the list  $L$ ;
  - 10: **end while**
  - 11: **for all** machine  $m_j$  **do**
  - 12:   Sort the tasks  $t_k$  on  $m_j$  in increasing  $ETC[t_k][m_j]$ ;
  - 13: **end for**
- 

Complexity :  $O(tm \log t)$

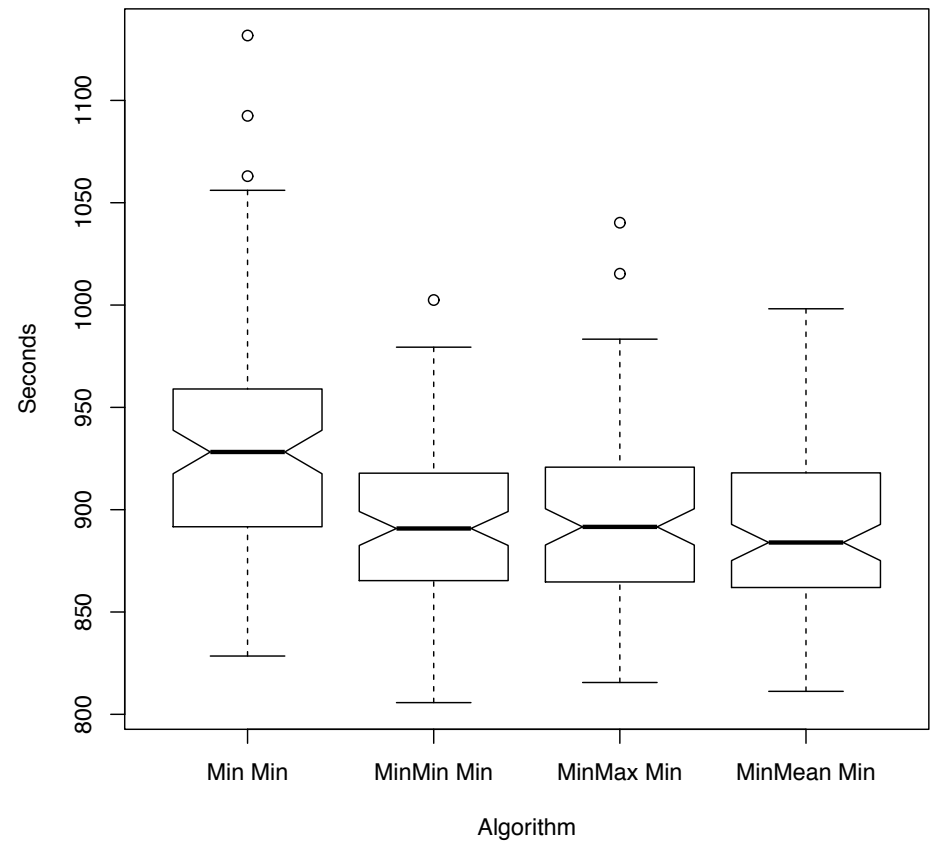
# Score Function

$$SF(t_i) = \lambda \cdot \frac{C_i}{\sum_{k=1}^m C_{ik}} + (1 - \lambda) \cdot \frac{ETC[t_i][m_j]}{\sum_{k=1}^m ETC[t_i][m_k]}, \quad (5)$$

Makespan



Makespan

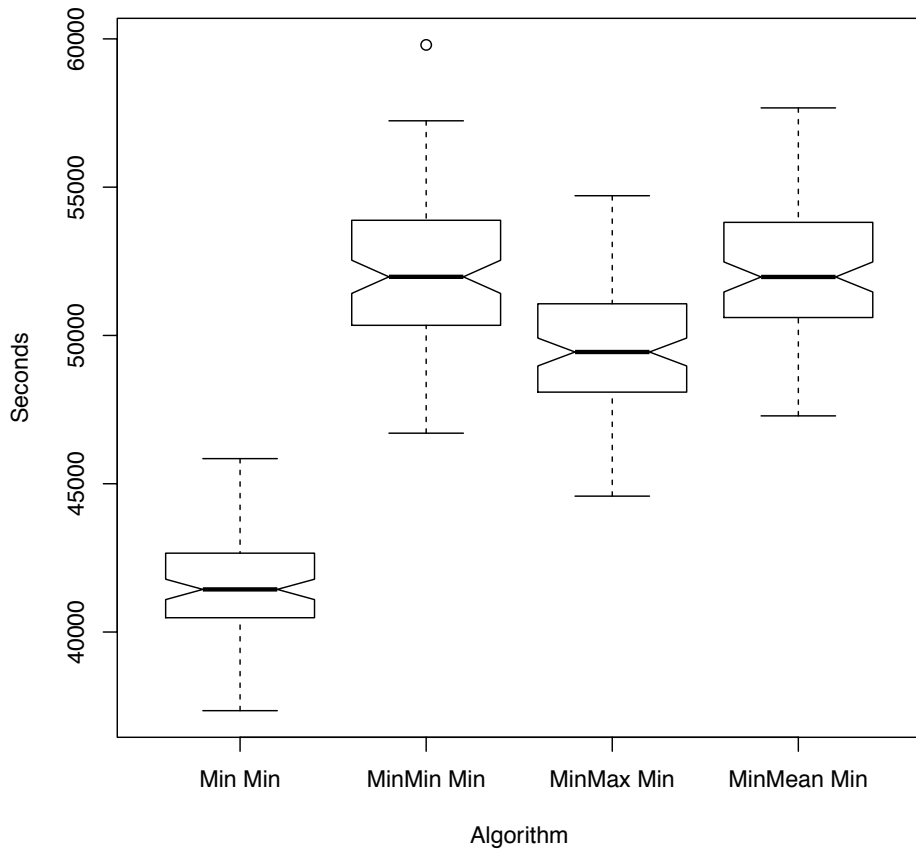


Lambda = 1

Lambda = 0.3

512 tasks x 16 machines, High task heterogeneity, High Machine Heterogeneity, Braun et al. Model

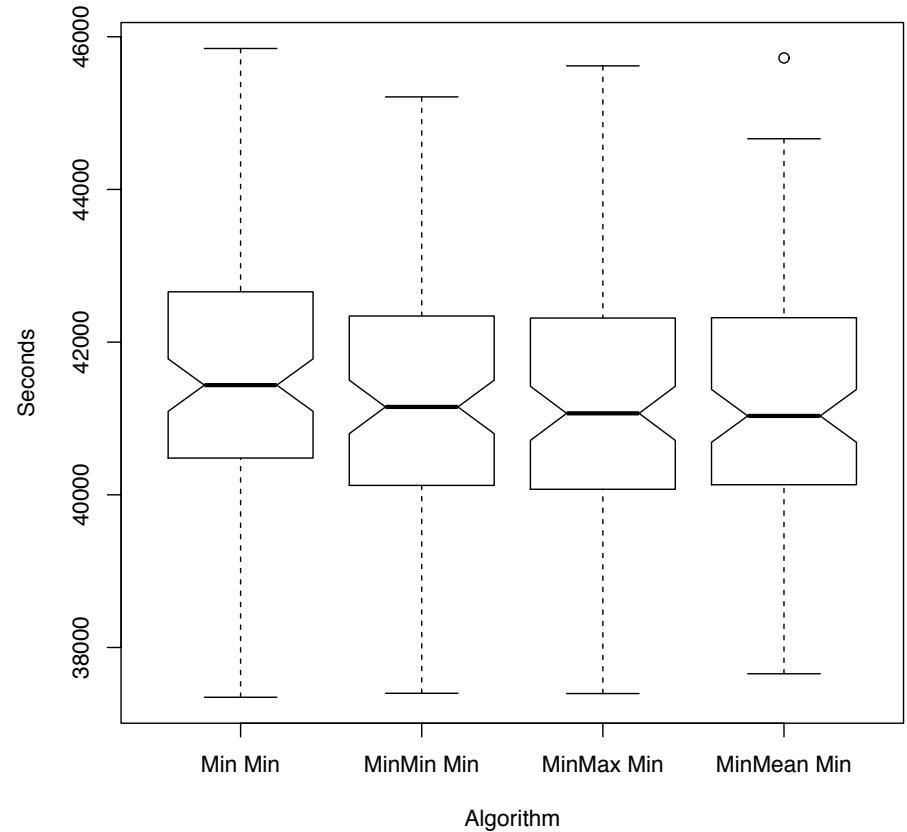
Real energy consumed before scaling



Lambda = 1

512 tasks x 16 machines

Real energy consumed before scaling



Lambda = 0.3

# Concluding remarks

- Energy efficiency is still an important issue in large-scale distributed systems
- Greening these systems involves many complex issues
- We investigated a software based approach
  - Energy Conscious Scheduling
- We designed a MO-solution based approach
- DVS has been adopted to contribute of the energy optimization
- Experiment results showed promising results.

# Concluding remarks

- Most of approaches consider best-effort approach, opportunistic computing can help
- Scheduling algorithms and policies can take advantage of monitoring and prediction
- Perspectives
  - To implement the algorithm in real time monitors
  - To combine proposed approach with DPM
  - To extend the model considering VMs

# Thank you for your attention!