Dynamické rozvrhování

Bc. Miroslava Plachá

Téma a motivace

- Rozvrh dynamicky se měnící množiny úloh na dynamicky se měnící množinu clusterů
- Výpočty v Metacentru
- Přístup pomocí programování s omezujícími podmínkami
- Integrace do simulátoru Alea

Použité nástroje I

- Choco
  - Systém v Javě pro programování s omezujícími podmínkami
  - Snadná definice omezení, heuristik výběru…
  - Systém model/solver

A + B*C = E

```
model.addConstraint(Choco.eq(E, Choco.plus(A, Choco.mult(B,C)))
```

Použité nástroje II

- Alea
  - simulátor gridu s výpadky
  - Java, GridSim
  - integrované algoritmy (FIFO, ESG, PBS…)
  - reálné sady úloh (Metacentrum…)
  - komunikace pomocí zpráv
  - neumožňuje rozvrhnutí úlohy paralelně mezi dva clustery

Definice problému

- Zdroje
  - jednotlivé clustery/stroje
  - charakteristika (lokace, síť, operační systém, licence…)
  - počet CPU
- Úlohy
  - uživatelem zadané programy
  - vyžadované parametry zdroje
  - vyžadovaný počet CPU
  - fronta
- Cíl: efektivní rozvržení úloh na zdroje

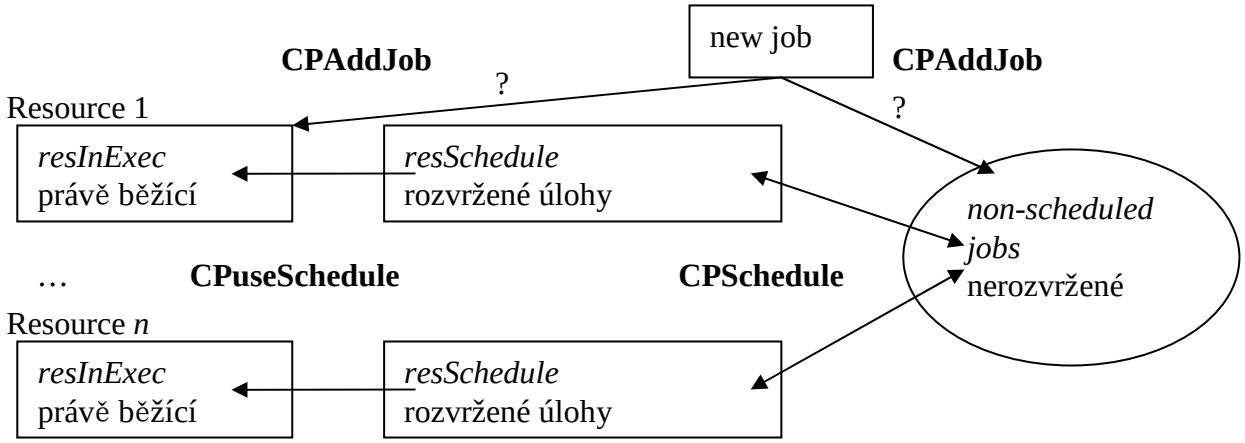
Optimalizační funkce

- non-delayed jobs
  - počet úloh které skončí do limitu
- tardiness
  - průměrná doba překročení limitu
- waiting time
  - start_time – arrival_time
- response time
  - finish_time – arrival_time
- slowdown
  - (finish_time – arrival_time)/cpu_time
- weighted slowdown (awsd)
  - (cpu_count* cpu_time) * slowdown
- weighted response (awrt)
  - (cpu_count * cpu_time) * response

Model - základní prvky

- Úloha (gridlet)
  - specifická množina požadavků
  - odhad délky trvání (deadline)
- Cluster (resource)
  - množina vlastností
  - počet CPU
  - seznamy resInExec a resSchedule

Struktura CP řešiče



- Seznam non-scheduled jobs
  - nerozvržené úlohy
- CPAddJob
  - metoda reagující na nově příchozí úlohu
- CPSchedule
  - vlastní rozvrhování pomocí kumulativního omezení
- CPuseSchedule
  - posílání úloh na stroje z resSchedule

Volání řešiče I

- Zpráva „Přijetí úlohy“
- Zpráva „Úloha dokončena“ – volný zdroj
- Zpráva „Selhání zdroje“
- Zpráva „Obnovení zdroje“
- Nejprve CPuseSchedule – maximalizace zatížení
- Podmíněno – minimalizace zbytečného rozvrhování
  - čas
  - záleží na zprávě

Volání řešiče II – rozbor podmínek

- **Deadlock** – detekována úloha, která nemůže být spuštěná
- **EmptyResSchedule**
  - nutné rozlišit nechtěný stroj od nevytíženého
  - Existuje resource *i* : *i*.resSchedule.size() == 0 AND existuje gridlet *j*: isSuitable(*i*,*j*).
- **Nová úloha** (pod limit)
  - CPSchedule
- **Volný zdroj**
  - CPuseSchedule
  - if (deadlock || EmptyResSchedule) CPSchedule
- **Obnovení stroje**
  - CPuseSchedule
  - if (CPuseSchedule == 0 || deadlock || EmptyResSchedule) CPSchedule
- **Selhání stroje**
  - CPuseSchedule
  - if (deadlock) CPSchedule

Algoritmus I

- resource_list – list of all resources
- sorted_resource_list – list of all resources sorted by the mips count
- resource.resSchedule – list of jobs scheduled for resource and currently not running
- resource.resInExec – list of jobs currently running on resource
- non_scheduled_jobs – list of non-scheduled jobs

- IF scheduler is currently not running
- THEN
- SWITCH message

- CASE „new job arrived“:
- IF size of non_scheduled_jobs > limit
- THEN
- CPAddJob new_job
- ELSE
- non_scheduled_jobs ← non_scheduled_jobs ∪ new_job
- CPSchedule
- FI
- break

Algoritmus II

- CASE „free resource“:
- CPuseSchedule
- IF deadlock OR (exists resource from resource_list: resource.resSchedule is empty
- AND exists job from non_scheduled_jobs: job can be run on resource)
- THEN
- CPSchedule
- FI
- break
-
- CASE „resource restart occured“:
- scheduled := CPuseSchedule
- IF deadlock OR scheduled == 0
- OR (exists resource from resource_list: resource.resSchedule is empty
- AND exists job from non_scheduled_jobs: job can be run on resource)
- THEN
- CPSchedule
- FI
- break
-

Algoritmus III

- CASE „resource failure occured“:
- CPuseSchedule
- IF deadlock
- THEN
- CPSchedule
- FI
- break
-
- DONE
- deadlock := false
- FI

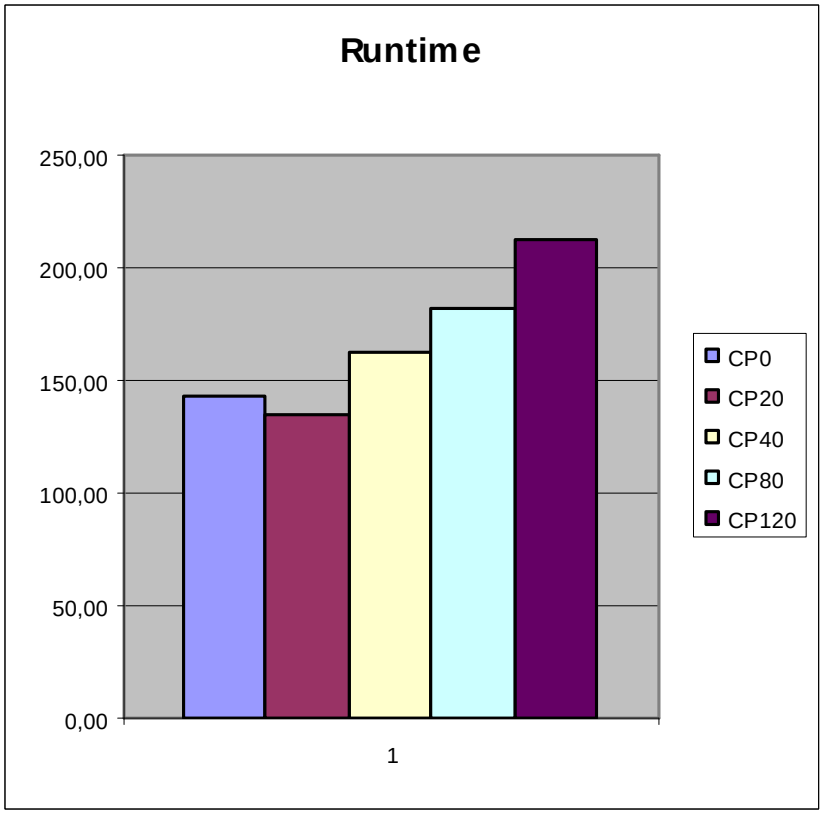
CPAddJob

- Možné scénáře
  - úlohu je možné ihned rozvrhnout – cena?
  - úlohu není možné rozvrhnout
- Dvojí přístup
  - resourceList = {1, 2, 3}, Dom(g_1)… Dom(g_{n-1}) = {1, 2}, Dom(g_n) = {1, 2, 3}
  - resSchedule.size() == 0
  - rozvrhnout jen pokud může být spuštěna vs zařadit do prázdné fronty
- Jak určit limit?

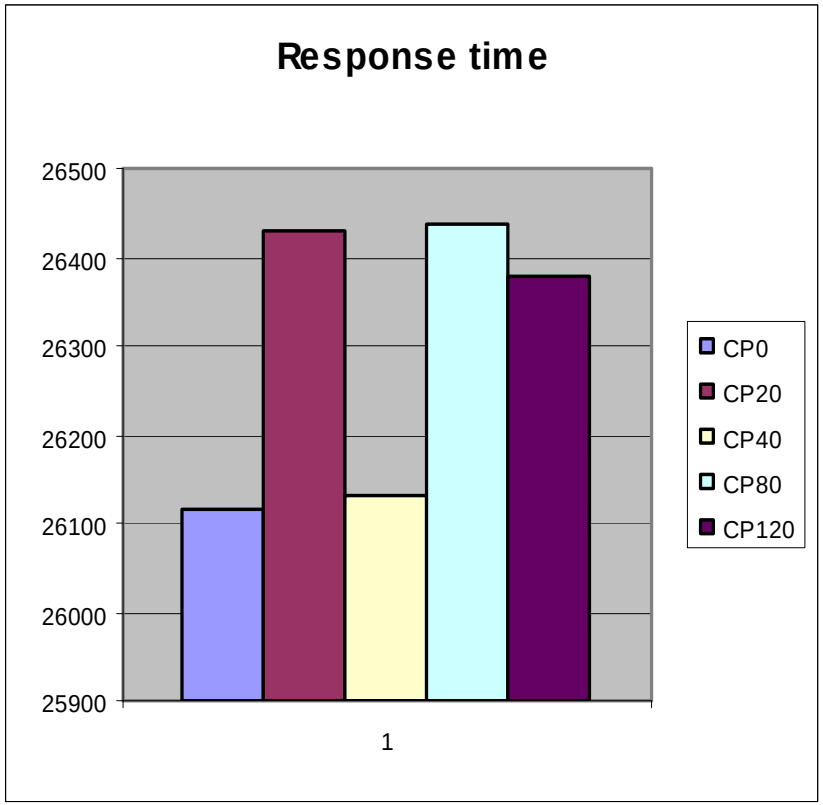
CPAddJob - algoritmus

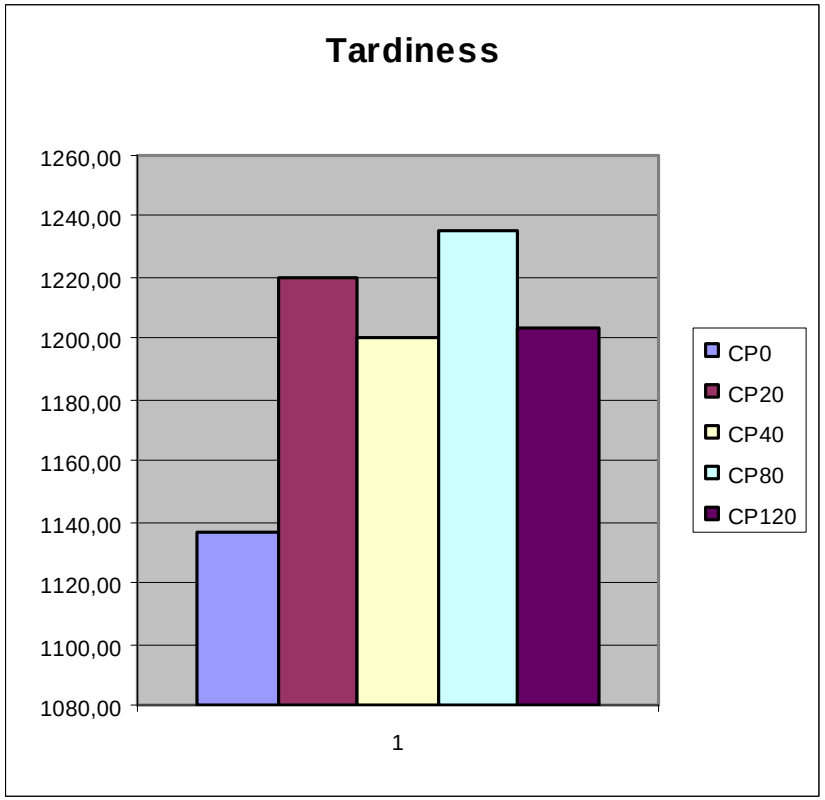
- job := new job
- IF size of non_scheduled_jobs > limit
- THEN
- FOR resource IN sorted_resource_list DO
- IF job can be currently assign to resource AND resource.resSchedule empty
- THEN
- resource.resInExec ← resource.resInExec ∪ job
- break
- FI
- DONE
-
- IF job was not assign
- THEN
- non_scheduled_jobs ← non_scheduled_jobs ∪ job
- FI
- FI

Experiment limit I



Experiment limit II



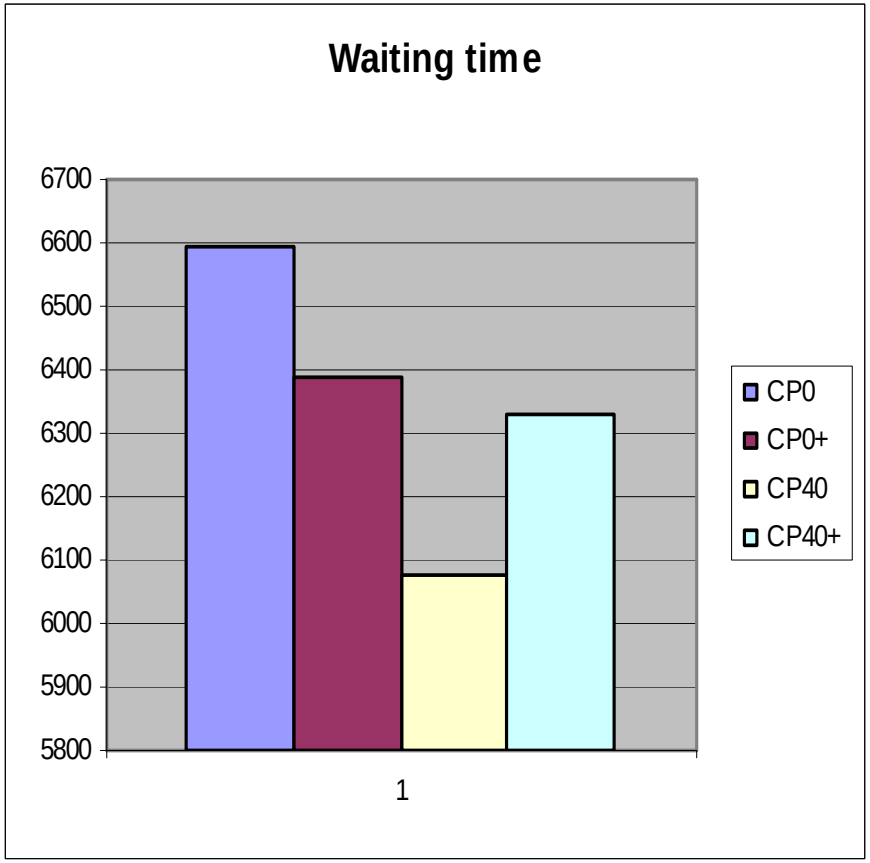


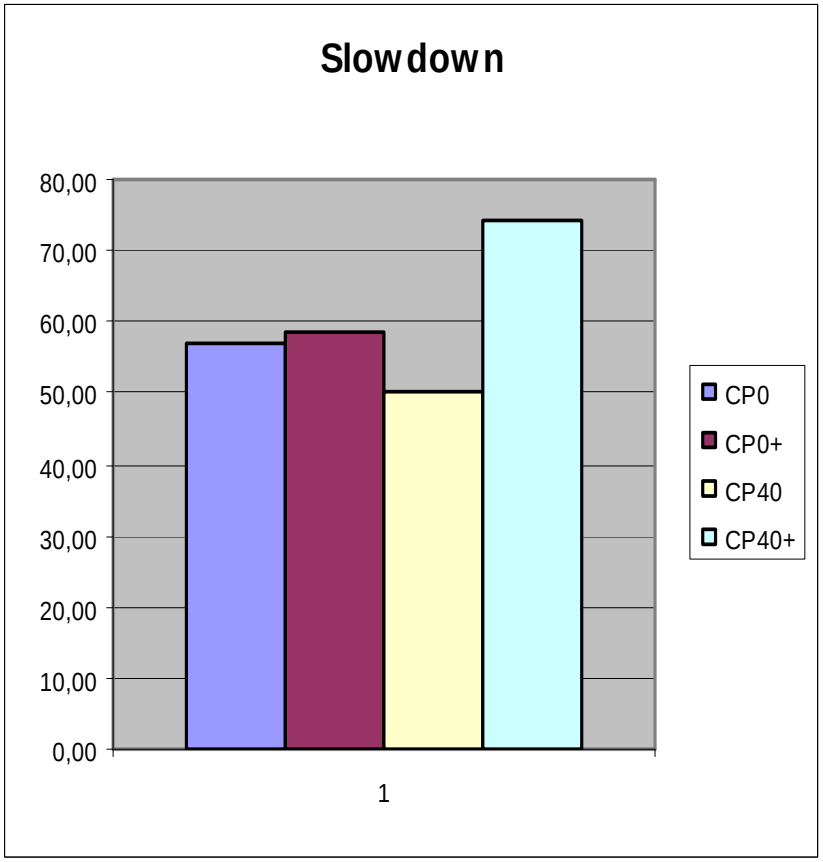
Jak daleko zajít?

- vkládání i do prázdného resSchedule – může mít za následek zhoršení, kterému by se předešlo rozvrhnutím
- cena vs efektivnost
- Konkrétní čísla v Metacentru:
  - celkem úloh 103655

Limit	Přímo resInExec	Přímo resSchedule	Ušetřeno rozvrhnutí
0	66%	2%	2%
40	9%	1,5%	2%

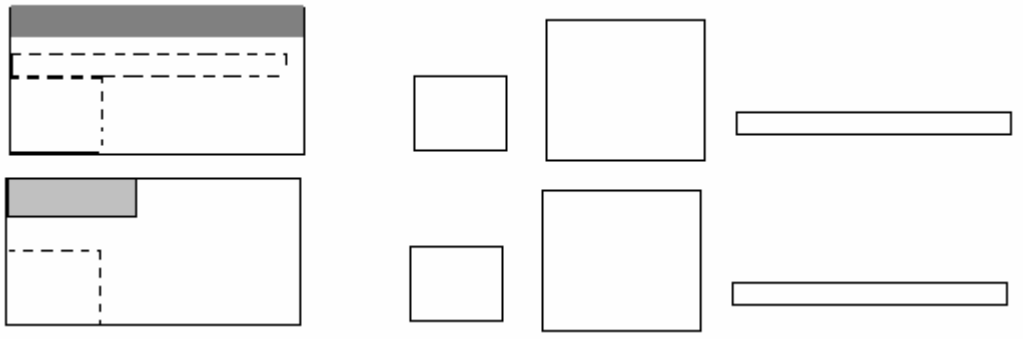
Jak daleko zajít? Experiment





CPuseSchedule

- posílání úloh na stroj v pořadí určeném řešičem.
- potenciální riziko – výpadek vs úmyslné rozvržení
  - přeskočení úlohy v případě výpadku



CPuseSchedule - algoritmus

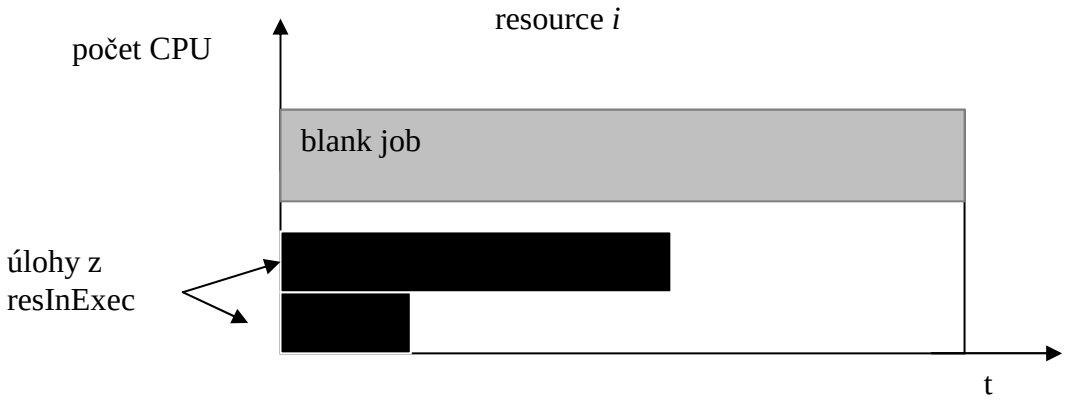
- scheduled := 0
- FOR resource IN resource_list DO
- WHILE next job in resource.resSchedule AND resource.free_CPUs > 0 DO
- IF next job in resource.resSchedule can be run
- THEN
- resource.resInExec ← resource.resInExec ∪ dequeue(resource.resSchedule)
- scheduled++
- ELSEIF next job in resource.resSchedule cannot be ever run
- THEN
- deadlock := true
- dequeue(resource.resSchedule)
- ELSE
- break
- FI
- DONE
- DONE
- RETURN scheduled

CPSchedule

- Určení množiny úloh pro rozvrhování
- Definice omezení
- Výpočet
- Seřazení úloh do resSchedule clusterů
- Zpráva volající CPuseSchedule – simulace časové náročnosti

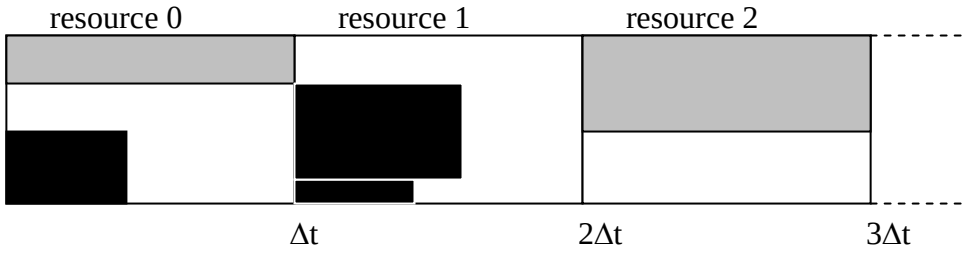
Soubor úloh pro rozvrhování

- Variabilní
  - seznam non-scheduled jobs
  - obsah resSchedule clusterů
- Pevné
  - blank jobs
  - resInExec clusterů



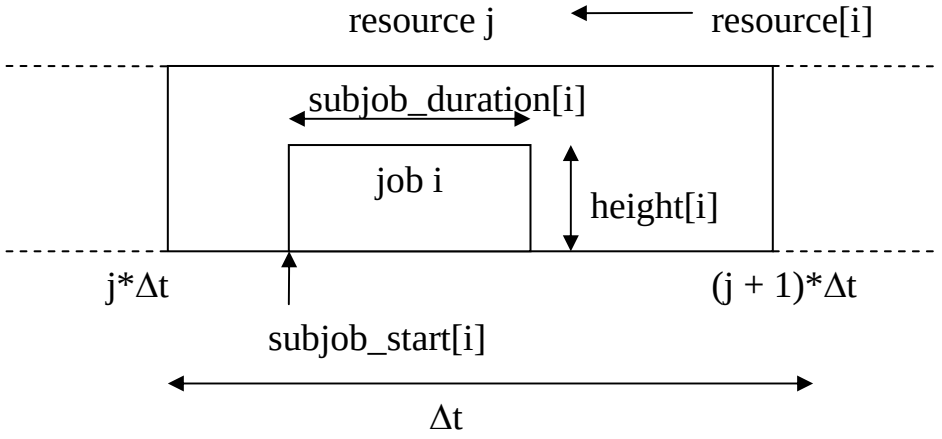
Proměnné I

- Δt – časový interval na který rozvrhujeme
- Jak dlouho bude úloha trvat?
- čas vs abstrakce (short = 1, normal = 2, long = 3, Δt = 4)
- Blank úloha *i* na clusteru *j*:
  - start[i] = j*Δt
  - height[i] = maxCPU – j.CPU
  - duration[i] = Δt
- resInExec úloha *i* na clusteru *j*:
  - start[i] = j*Δt
  - height[i] = i.CPU
  - duration[i] = ?



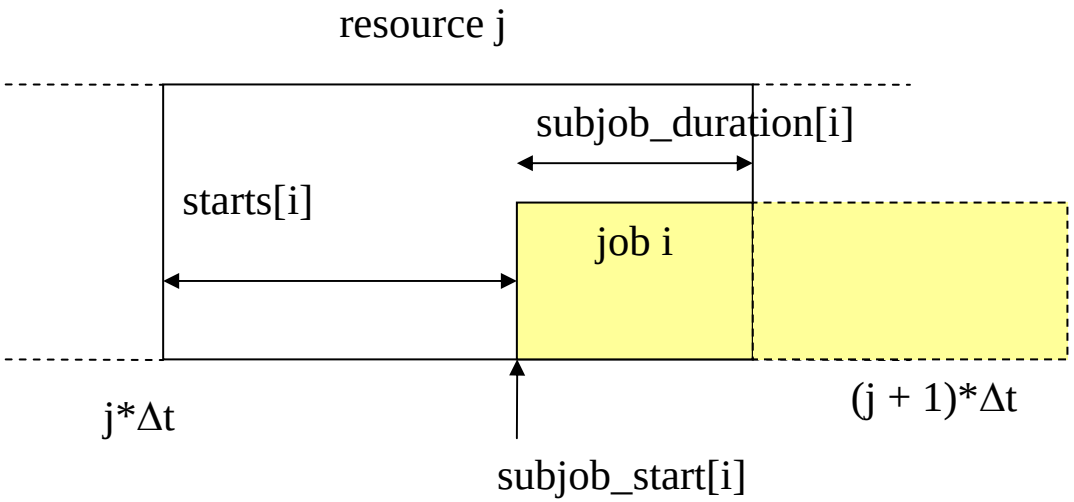
Proměnné II

- Nerozvržená úloha *i*:
  - starts[i]; Dom(starts[i]) = 0 .. Δt
  - height[i] = i.CPU
  - resource[i]; Dom(resource[i]) = vyhovující stroje
    - enum
  - subjob_start[i]; Dom(subjob_start[i]) = 0 .. resList.size() * Δt
  - subjob_duration[i];
    Dom(subjob_duration[i]) = 0 .. min(Δt, i.duration)



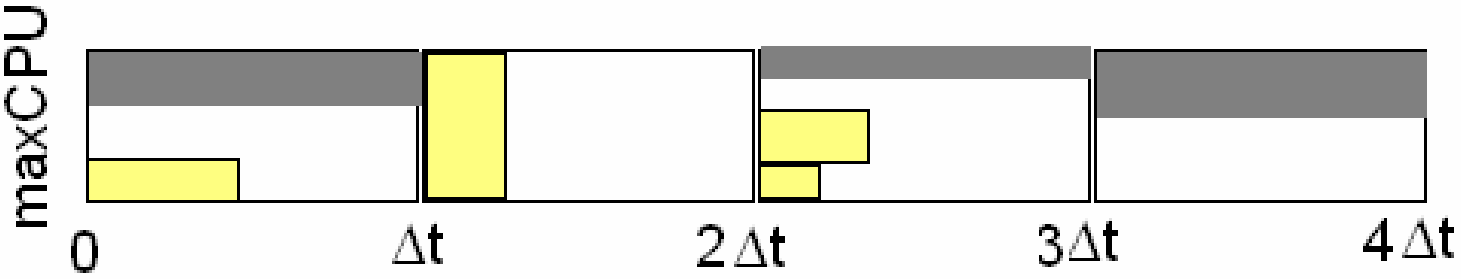
Omezení

- Nerozvržená úloha *i*:
  - subjob_start[i] = Δt * resource[i] + starts[i]
  - subjob_duration[i] = min(Δt - starts[i], min (i.duration, Δt))
    - dovoluje rozvrhnout jen začátek úlohy
    - nerozvrhnuté úlohy se „schovají“ v délce trvání 0



Kumulativní omezení

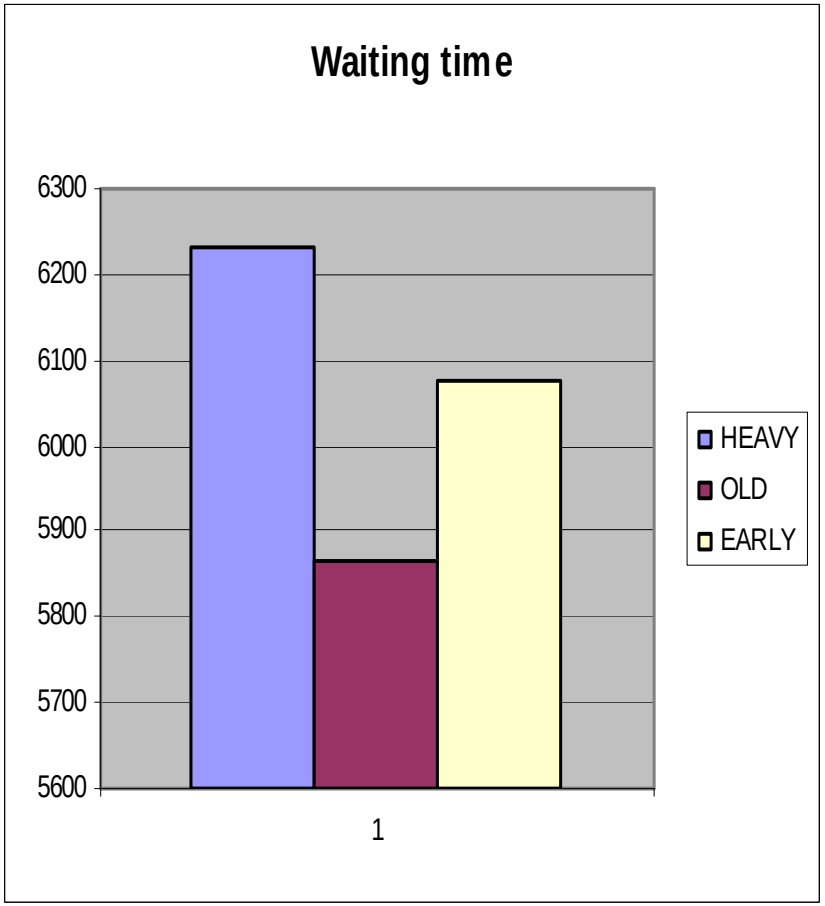
- cumulative(starts, durations, heights, maxCPU)
- ve starts – start[i] pro blank a resInExec úlohy, subjob_start pro nerozvrhnuté
- Jak moc do budoucnosti je efektivní rozvrhovat? Příliš mnoho úloh - zpomalení

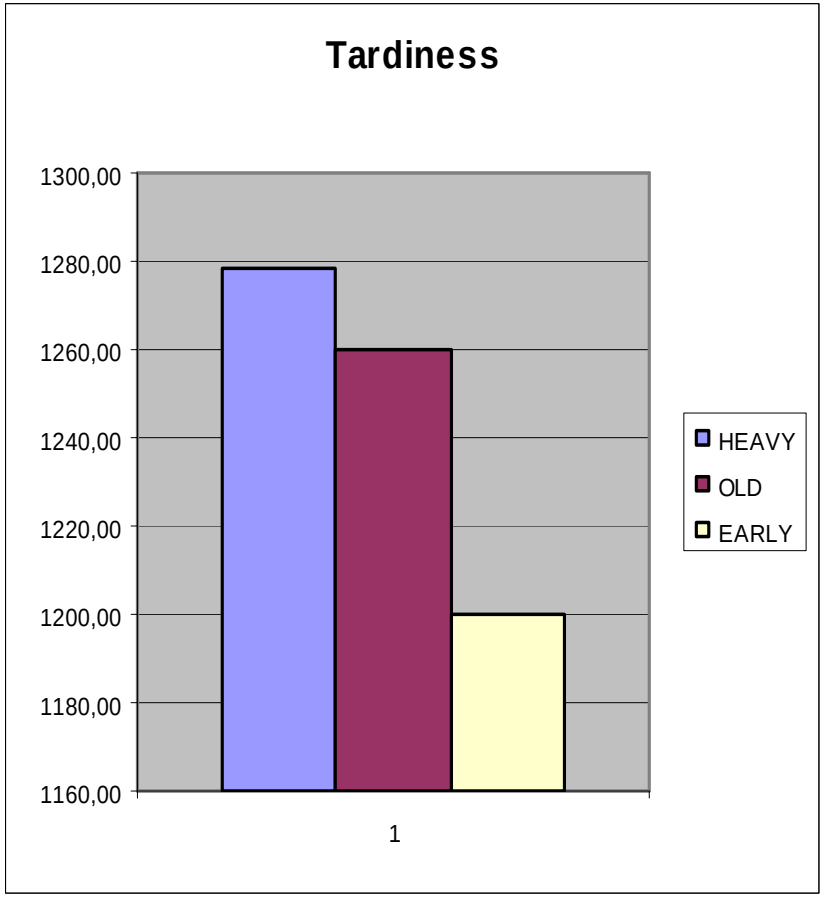


Hledání řešení - heuristiky

- při hledání řešení se pouze přiřazují hodnoty proměnným resource[i] (enum) a starts[i] (definováno mezemi)
- heuristika výběru proměnné – které *i* jako první?
  - EarliestGridlet – preferuje v minulosti již přiřazené
  - OldestGridlet – preferuje nejstarší
  - HeaviestGridlet – preferuje „největší“
  - SmallestDomain – preferuje nejvíce náročnou
- heuristika výběru hodnoty
  - starts[i] - IncreasingDomain – co nejmenší čas
  - resource[i] – co nejlukrativnější?
    - MinimalUsage – FreeCPU/(InExec.size() + usage)
      - pokud shoda – náhodně

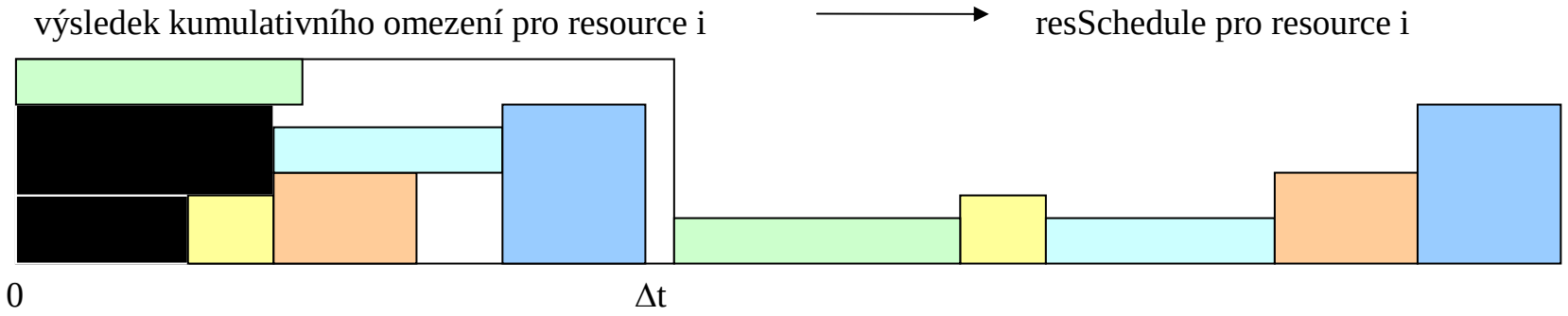
Experiment - heuristiky





Hledání řešení - výsledek

- Seznam úloh s přiřazenou začáteční dobou < Δt
- Chronologické seřazení
- Seřazení dle výšky (nahrazuje 2D přístup)
- Zapsání do fronty resSchedule příslušného clusteru
- Nepřiřazené úlohy zůstávají v non-scheduled jobs



Optimalizace

- Cena?
- Limit úloh pro provedení?
- Optimalizační funkce?

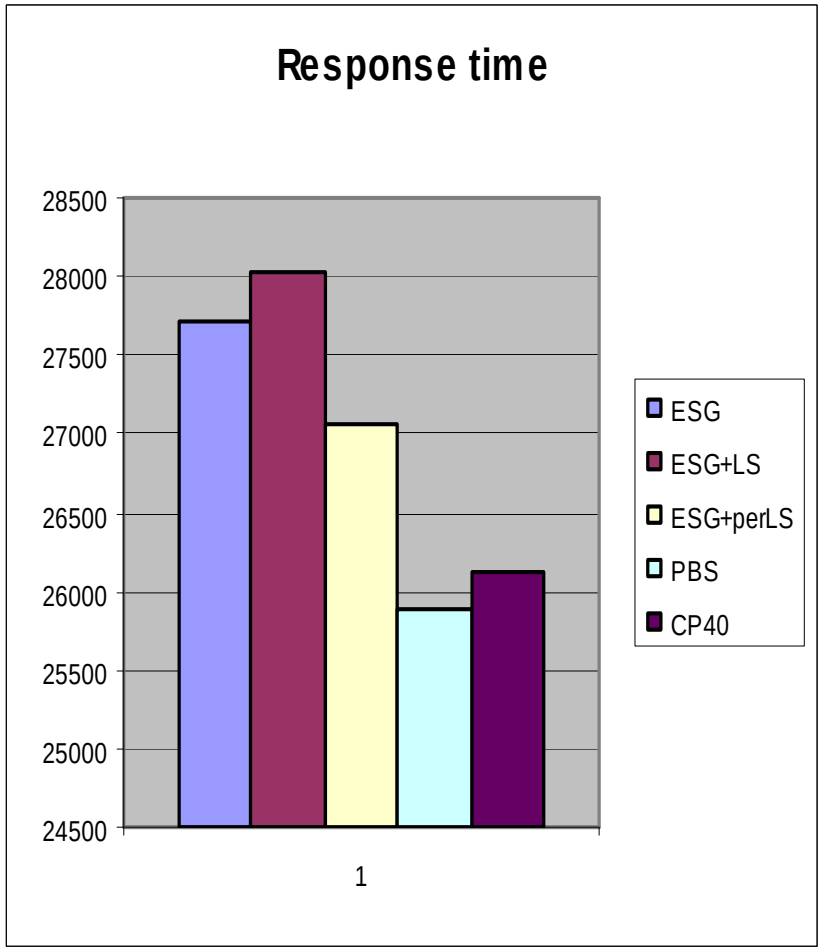
Riziko nezachycení zprávy

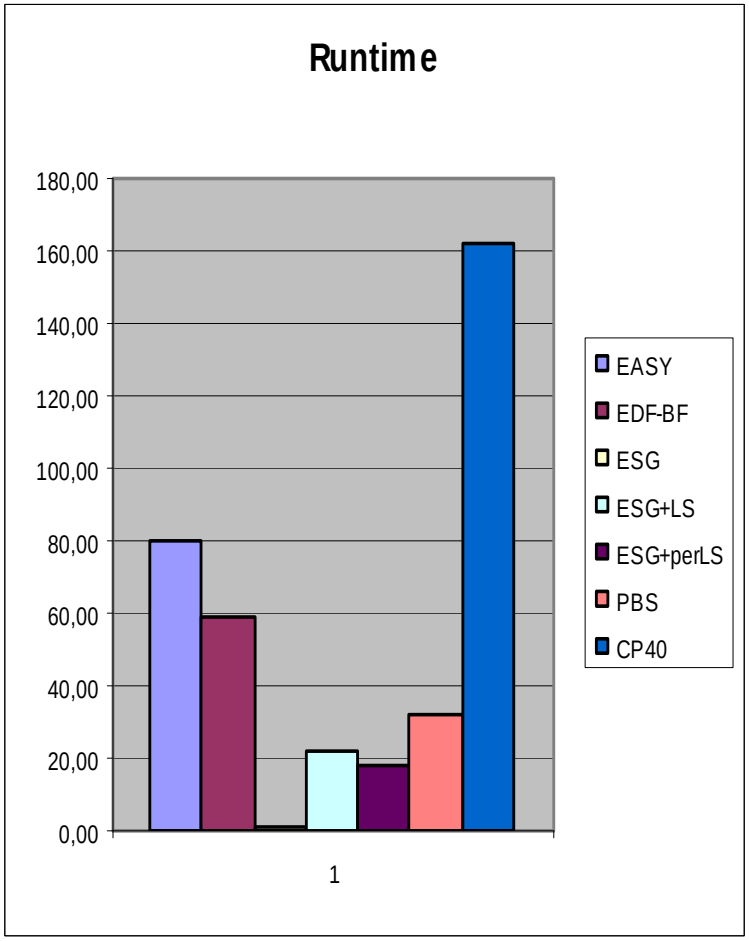
- Rozvrhování trvá *x* sec a v čase *t* < *x* přijde zpráva
- Volný stroj – zpožděná reakce *x* – *t* sekund (CPuseSchedule bude voláno)
- Obnovení stroje – dtto
- Selhání stroje – detekce uvázlé úlohy, jinak OK
- Nová úloha – nemůžeme kvůli probíhajícímu rozvrhování rozvrhnout hned – do příštího rozvrhování možná neefektivnost

Časová náročnost

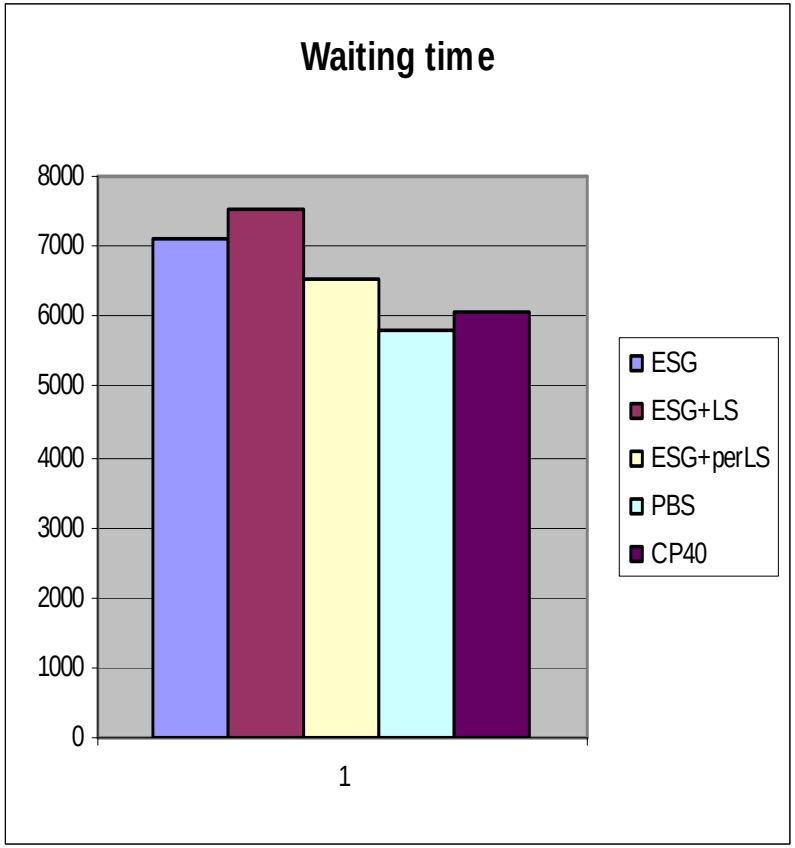
	Celkem	Pod 1s	1s – 30s	déle	maximum
CP0	16 295	85,8%	14,2%	(17)	100 s
CP40	64 548	95,2%	4,8%	(32)	116 s
CPmax	78 333	93,2%	6,7%	(80)	74 s

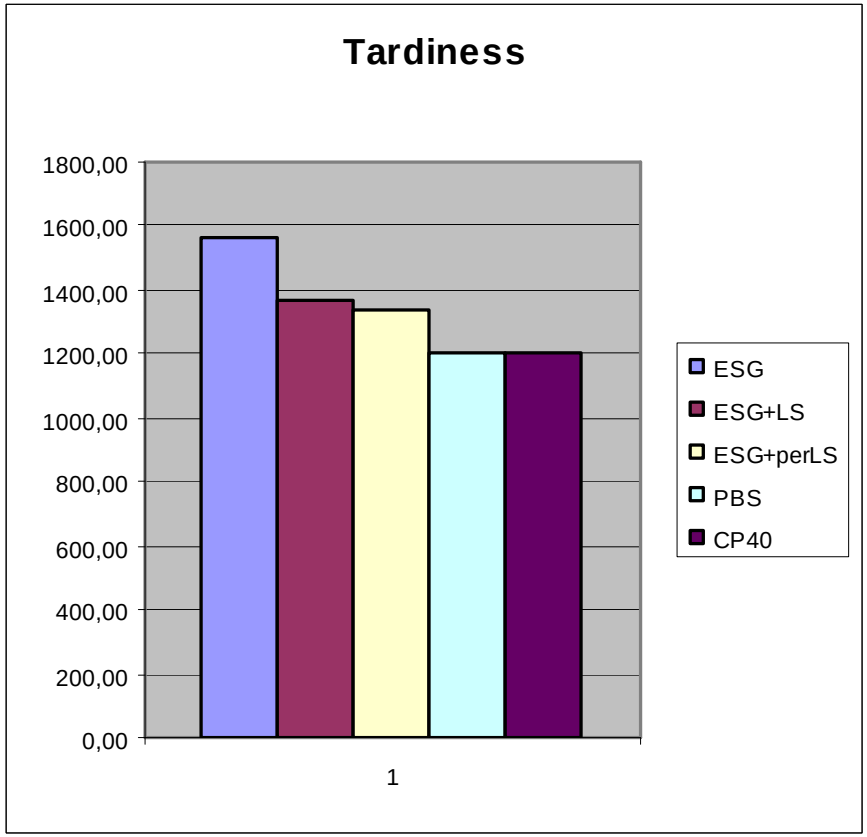
Experimenty – srovnání I



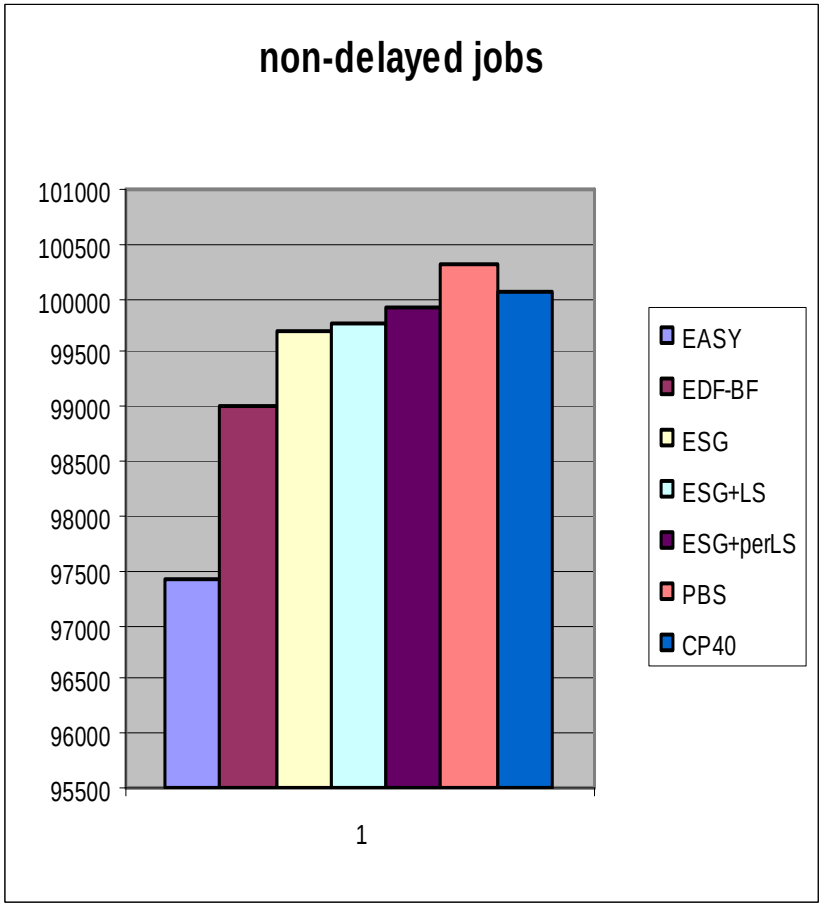


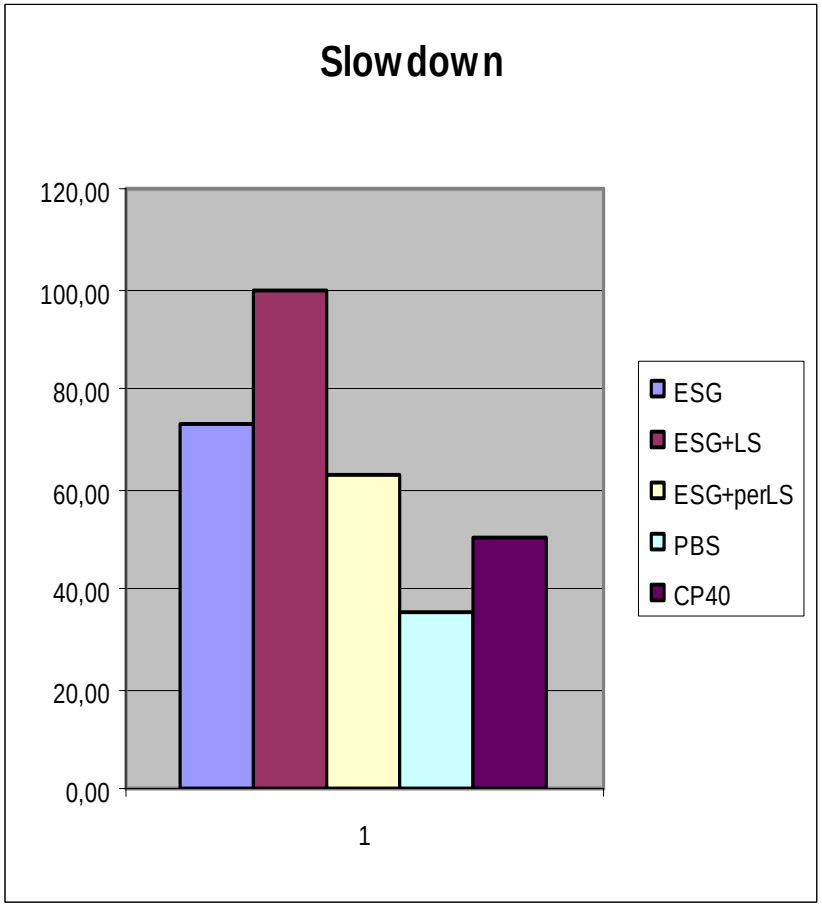
Experimenty – srovnání II



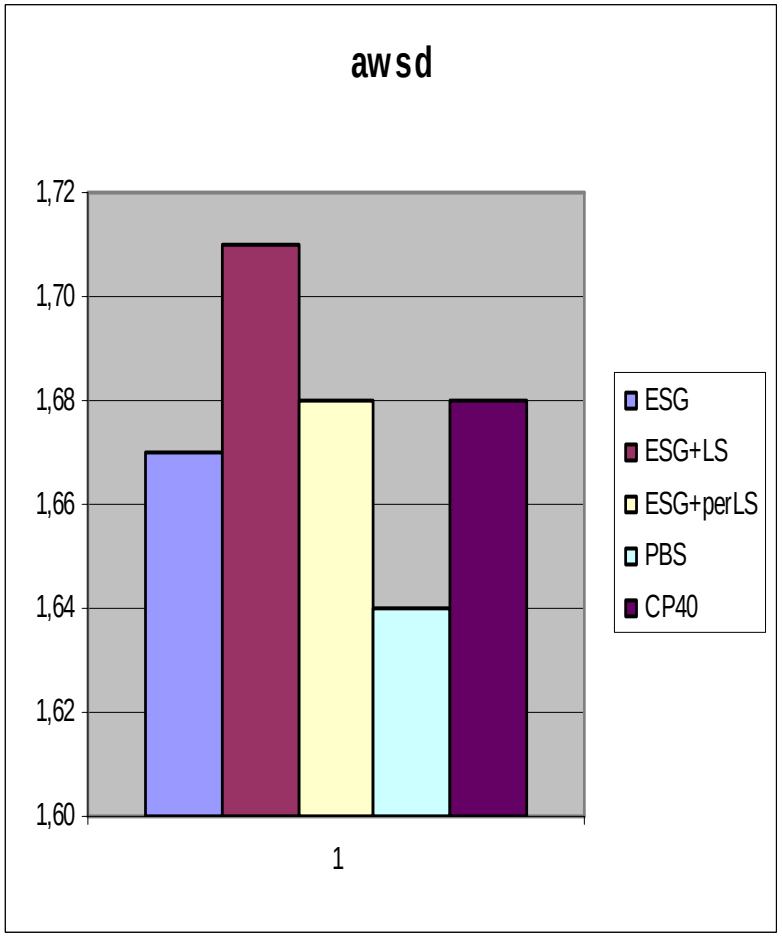


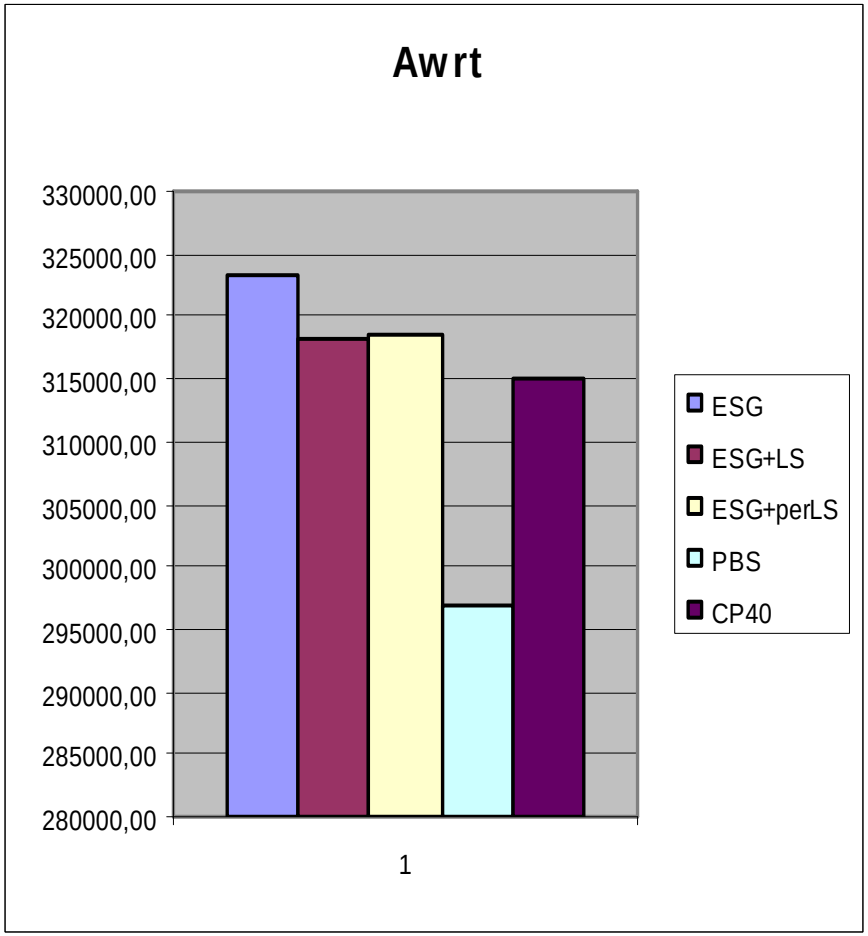
Experimenty – srovnání II





Experimenty – srovnání III





Výhody/nevýhody

- + Snadná definice chytrých heuristik pro výběr proměnných a hodnot
- Jak určit univerzální heuristiku pro nejrůznější typy sad úloh?
- + Oproti frontovým algoritmům pohled na úlohy jako celek a přehodnocování jejich pořadí
- + Jednoduché rozšíření funkčnosti i pro rozvrhování na paralelní clustery
- + Jednoduché přidávání dalších omezení (počet úloh na uživatele…)
- - časová a paměťová náročnost – riziko zahlcení
  - lze detekovat a přejít na jiný, méně náročný algoritmus
- + zvyšování efektivnosti použitím lepšího hardware (-)
- - nedeterministické
- - neodhadnutelné chování