Bi-objective Scheduling Algorithms for Optimizing Makespan and Reliability on Heterogeneous Systems

Jack J. Dongarra, **Emmanuel Jeannot** and Zhiao Shi

INRIA & Innovative Computing Laboratory
dongarra@cs.utk.edu, ejeannot@loria.fr, shi@cs.utk.edu

Outline of the talk

Introduction

Problem studied:

- scheduling DAG
- heterogeneous systems
- hardware can fail

Bi-criteria objective:

- given a makespan objective
- optimize reliability

Related work

A "*new subject*" :

- Dogan & Ozgüner 2002: Model the problem, RDLS bi-criteria heuristic.
- Dogan & Ozgüner 2004: enhancement of previous result (GA).
- Qin & Jiang 2005: first optimize deadline, then maximize reliability.
- Hakem & Butelle 2006: BSA, bi-criteria heuristic that outperforms RDLS.

Modeling

- $G = (V, E)$: a DAG.
- $v_i \in V$ is associated a number of operations: o_i.
- $n = |V|$
- $e_i = (i, j) \in E$ is associated l_i the time to send data from task v_i to task v_j (if they are not executed on the processor).
- a set P of m processors
- processor $p_j \in P$ is associated with two values:
  - τ_j the time to perform one operation and
  - λ_j the failure rate.
- v_i executed on p_j will last $o_i \times \tau_j$.

Modeling

- $G = (V, E)$: a DAG.
- $v_i \in V$ is associated a number of operations: o_i.
- $n = |V|$
- $e_i = (i, j) \in E$ is associated l_i the time to send data from task v_i to task v_j (if they are not executed on the processor).
- a set P of m processors
- processor $p_j \in P$ is associated with two values:
  - τ_j the time to perform one operation and
  - λ_j the failure rate.
- v_i executed on p_j will last $o_i \times \tau_j$.

Assumption:

- During the execution of the DAG, the failure rate is constant.

$\Rightarrow$ failure model follows an exponential law.

$\Rightarrow$ probability that v_i finishes (correctly) its execution:

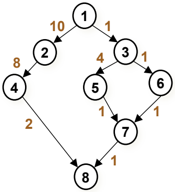
$$e^{-o_i \times \tau_j \times \lambda_j}$$

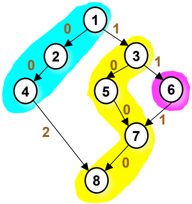
Outline

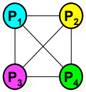
Scheduling problem

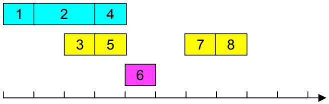
Allocate tasks to processors such that:

- two tasks cannot be allocated to the same processor at the same time,
- dependencies are respected.









Criteria

C_j: termination date of processor j

Two criteria to optimize:

- **Makespan**: minimize

$$M = \max(C_j)$$

- **Reliability**: maximize

$$p_{\text{succ}} = \prod_{j=1}^{m} e^{-C_j \lambda_j} = e^{-\sum_{j=1}^{m} C_j \lambda_j}$$

  or minimize

$$\sum_{j=1}^{m} C_j \lambda_j$$

Two unrelated criteria

Proposition

Let S be a schedule where all the tasks have been assigned, in topological order, to the processor i such that $\lambda_i \tau_i$ **is minimum**. *Then any schedule S' is such that $p'_{succ} \leq p_{succ}$.*

Two unrelated criteria

Proposition

Let S be a schedule where all the tasks have been assigned, in topological order, to the processor i such that $\lambda_i\tau_i$ **is minimum**. *Then any schedule S' is such that $p'_{succ} \leq p_{succ}$.*

Proof

- s.w.l.o.g $i = 1$ ($i.$ $e.$, $\forall j:\ \tau_1\lambda_1 \leq \tau_j\lambda_j$).
- $p_{\text{succ}} = e^{-C_1\lambda_1}$, $p'_{\text{succ}} = e^{-\sum_{j=0}^{m} C'_j\lambda_j}$.
- $T = T_2 \cup \ldots \cup T_m$, sets of the tasks allocated to processors $2, \ldots, m$ by S'.
- $C'_1 \geq C_1 - \tau_1 \sum_{v_i \in T} o_i$.
- $\forall\, 2 \leq j \leq m,\ C'_j \geq \tau_j \sum_{v_i \in T_j} o_i$

Two unrelated criteria

Proposition

Let S be a schedule where all the tasks have been assigned, in topological order, to the processor i such that $\lambda_i\tau_i$ **is minimum**. *Then any schedule S' is such that $p'_{succ} \leq p_{succ}$.*

Proof

- s.w.l.o.g $i = 1$ (*i. e.*, $\forall j: \ \tau_1\lambda_1 \leq \tau_j\lambda_j$).
- $p_{\text{succ}} = e^{-C_1\lambda_1}$, $p'_{\text{succ}} = e^{-\sum_{j=0}^{m} C'_j\lambda_j}$.
- $T = T_2 \cup \ldots \cup T_m$, sets of the tasks allocated to processors $2, \ldots, m$ by S'.
- $C'_1 \geq C_1 - \tau_1 \sum_{v_i \in T} o_i$.
- $\forall\, 2 \leq j \leq m, \ \ C'_j \geq \tau_j \sum_{v_i \in T_j} o_i$

$$\sum_{j=1}^{m} C'_j\lambda_j - C_1\lambda_1 \quad \geq \quad \sum_{j=2}^{m} \left((\tau_j\lambda_j - \tau_1\lambda_1) \sum_{v_i \in T_j} o_i \right) \geq 0$$

$$\Rightarrow \frac{p_{\text{succ}}}{p'_{\text{succ}}} = e^{\sum_{j=1}^{m} C'_j\lambda_j - C_1\lambda_1} \geq 1$$

Bi-criteria scheduling

Objective: maximizing the reliability subject to the condition that the makespan is minimized.

- Finding the optimal makespan, is most of the time NP-hard,
- we aim at designing an $(\alpha,\ \beta)$-approximation algorithm.
- $(\alpha,\ \beta)$-approximation algorithm:
  - makespan at most α times larger than the optimal one,
  - probability of failure is at most β times larger than the optimal one (among the schedules that minimize the makespan).

Approximation algorithm and probability

Let p_{succ} (resp. p_{fail}) be the probability of success (resp. of failure) of a schedule S.

Let $\tilde{p}_{\text{succ}}$ (resp. $\tilde{p}_{\text{fail}}$) be the optimal probability of success (resp. of failure) for the same input as S.

Approximation algorithm and probability

Let p_{succ} (resp. p_{fail}) be the probability of success (resp. of failure) of a schedule S.
Let $\tilde{p}_{\text{succ}}$ (resp. $\tilde{p}_{\text{fail}}$) be the optimal probability of success (resp. of failure) for the same input as S.

$$\beta = 5\,, \tilde{p}_{\text{fail}} = 0.3 \;\Rightarrow\; p_{\text{fail}} \leq \beta \cdot \tilde{p}_{\text{fail}} = 5 \times 0.3 = 1.5!$$

Approximation algorithm and probability

Let p_{succ} (resp. p_{fail}) be the probability of success (resp. of failure) of a schedule S.

Let $\tilde{p}_{\text{succ}}$ (resp. $\tilde{p}_{\text{fail}}$) be the optimal probability of success (resp. of failure) for the same input as S.

$$\beta = 5\,,\tilde{p}_{\text{fail}} = 0.3 \;\Rightarrow\; p_{\text{fail}} \leq \beta \cdot \tilde{p}_{\text{fail}} = 5 \times 0.3 = 1.5!$$

Proposition

$$p_{succ} \geq \tilde{p}_{succ}^{\beta} \Rightarrow p_{fail} \leq \beta \cdot \tilde{p}_{fail}$$

Approximation algorithm and probability

Let p_{succ} (resp. p_{fail}) be the probability of success (resp. of failure) of a schedule S.
Let $\tilde{p}_{\text{succ}}$ (resp. $\tilde{p}_{\text{fail}}$) be the optimal probability of success (resp. of failure) for the same input as S.

$$\beta = 5\,, \tilde{p}_{\text{fail}} = 0.3 \Rightarrow p_{\text{fail}} \leq \beta \cdot \tilde{p}_{\text{fail}} = 5 \times 0.3 = 1.5!$$

Proposition

$$p_{succ} \geq \tilde{p}_{succ}^{\beta} \Rightarrow p_{fail} \leq \beta \cdot \tilde{p}_{fail}$$

Proof The proof is based on the Taylor's series of $(1-x)^n$, where, $\forall x \in [0,1], \forall n \in [1, +\infty[, (1-x)^n \leq 1 - nx$.

$$\begin{aligned} p_{\text{fail}} &= 1 - p_{\text{succ}} \leq 1 - \tilde{p}_{\text{succ}}^{\beta} = 1 - (1 - \tilde{p}_{\text{fail}})^{\beta} \\ &\leq 1 - (1 - \beta \cdot \tilde{p}_{\text{fail}}) = \beta \cdot \tilde{p}_{\text{fail}} \end{aligned}$$

Outline

Independent unitary tasks

$o_i = 1$ and $E = \emptyset$, $n = |V|$.

Independent unitary tasks

$o_i = 1$ and $E = \emptyset$, $n = |V|$.

Algorithm 1 Makespan-optimal allocation for independent unitary tasks

for i=1 to P
  $n_i \leftarrow \left\lfloor \frac{1/\tau_i}{\sum 1/\tau_i} \right\rfloor \times n$
while $\sum n_i < n$
  $k = \text{argmin}(\tau_k(n_k + 1))$
  $n_k \leftarrow n_k + 1$

Independent unitary tasks

$o_i = 1$ and $E = \emptyset$, $n = |V|$.

Algorithm 1 Makespan-optimal allocation for independent unitary tasks

for i=1 to P

$n_i \leftarrow \left\lfloor \frac{1/\tau_i}{\sum 1/\tau_i} \right\rfloor \times n$

while $\sum n_i < n$

$k = \text{argmin}(\tau_k(n_k + 1))$

$n_k \leftarrow n_k + 1$

Above algorithm gives M_{opt} the best achievable makespan.
For the reliability criteria the user gives the value of α that tells how far from the optimal makespan he/she can tolerate to be.
Then we compute a schedule such that:

- $M \leq \alpha M_{\text{opt}}$
- it has the best reliability among all the schedules with makespan $\leq M$.

Optimal algorithm for Independent unitary tasks

Algorithm 2 Optimal reliable allocation for independent unitary tasks

Input: $\alpha \in [1, +\infty[$

Compute $M = \alpha M_{\text{opt}}$ using previous algorithm

Sort the processor by increasing $\lambda_i \tau_i$

$X \leftarrow 0$

for i=1 to P

  if $X < N$

    $n_i \leftarrow \min\left(N - X, \left\lfloor \frac{M}{\tau_i} \right\rfloor\right)$

  else

    $n_i \leftarrow 0$

  $X \leftarrow X + n_i$

Proof of optimality of the reliability

We need to show that $\sum_{i \in [1,P]} n_i \lambda_i \tau_i$ is minimum.

Proof of optimality of the reliability

We need to show that $\sum_{i \in [1,P]} n_i \lambda_i \tau_i$ is minimum.

- First let us remark that the algorithm fills the processor of task in the increasing order of $\lambda_i \tau_i$.

Proof of optimality of the reliability

We need to show that $\sum_{i \in [1,P]} n_i \lambda_i \tau_i$ is minimum.

- First let us remark that the algorithm fills the processor of task in the increasing order of $\lambda_i \tau_i$.

$\Rightarrow$ any other valid allocation $\{n'_1, \ldots, n'_N\}$ is such that $n'_i < n_i$ and $n'_j > n_j$ for any $i < j$.

Proof of optimality of the reliability

We need to show that $\sum_{i\in[1,P]} n_i\lambda_i\tau_i$ is minimum.

- First let us remark that the algorithm fills the processor of task in the increasing order of $\lambda_i\tau_i$.

$\Rightarrow$ any other valid allocation $\{n'_1,\dots,n'_N\}$ is such that $n'_i < n_i$ and $n'_j > n_j$ for any $i < j$.

- w.l.o.g. let $n'_1 = n_1 - k$, $n'_i = n_i + k$ and $n'_j = n_j$ for $k \in [1, n_i]$, $j \neq 1$ and $j \neq i$.

Proof of optimality of the reliability

We need to show that $\sum_{i \in [1,P]} n_i \lambda_i \tau_i$ is minimum.

- First let us remark that the algorithm fills the processor of task in the increasing order of $\lambda_i \tau_i$.

$\Rightarrow$ any other valid allocation $\{n'_1, \ldots, n'_N\}$ is such that $n'_i < n_i$ and $n'_j > n_j$ for any $i < j$.

- w.l.o.g. let $n'_1 = n_1 - k$, $n'_i = n_i + k$ and $n'_j = n_j$ for $k \in [1, n_i]$, $j \neq 1$ and $j \neq i$.
- Then the difference between the two objective values is:

Proof of optimality of the reliability

We need to show that $\sum_{i\in[1,P]} n_i\lambda_i\tau_i$ is minimum.

- First let us remark that the algorithm fills the processor of task in the increasing order of $\lambda_i\tau_i$.

$\Rightarrow$ any other valid allocation $\{n'_1,\ldots,n'_N\}$ is such that $n'_i < n_i$ and $n'_j > n_j$ for any $i < j$.

- w.l.o.g. let $n'_1 = n_1 - k$, $n'_i = n_i + k$ and $n'_j = n_j$ for $k \in [1, n_i]$, $j \neq 1$ and $j \neq i$.
- Then the difference between the two objective values is:

$$\begin{aligned}
X &= n_1\lambda_1\tau_1 + \ldots + n_i\lambda_i\tau_i + \ldots + n_N\lambda_N\tau_N - n'_1\lambda_1\tau_1 - \ldots - n'_i\lambda_i\tau_i - \ldots + n'_N\lambda_N\tau_N \\
&= \lambda_1\tau_1(n_1 - n'_1) + \lambda_i\tau_i(n_i - n'_i) \\
&= k\lambda_1\tau_1 - k\lambda_i\tau_i \\
&= k(\lambda_1\tau_1 - \lambda_i\tau_i) \\
&\leq 0 \text{ because } \lambda_i\tau_i \geq \lambda_1\tau_1.
\end{aligned}$$

Hence, the first allocation has a smaller objective value.

Outline

Independent tasks: the makespan problem

$E = \emptyset$

Independent tasks: the makespan problem

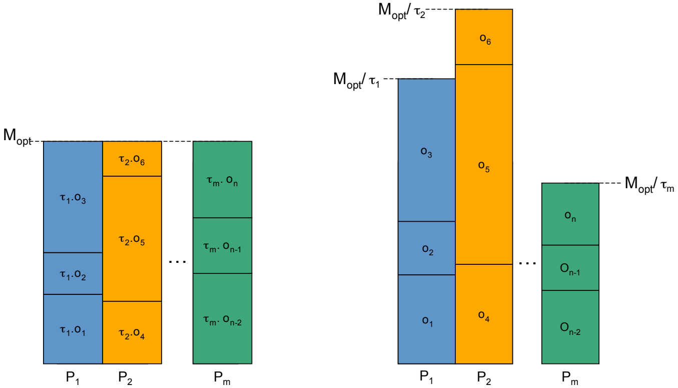
$E = \emptyset$

Makespan problem related to the 1-D bin-packing problem with variable bin size.

Independent tasks: the makespan problem

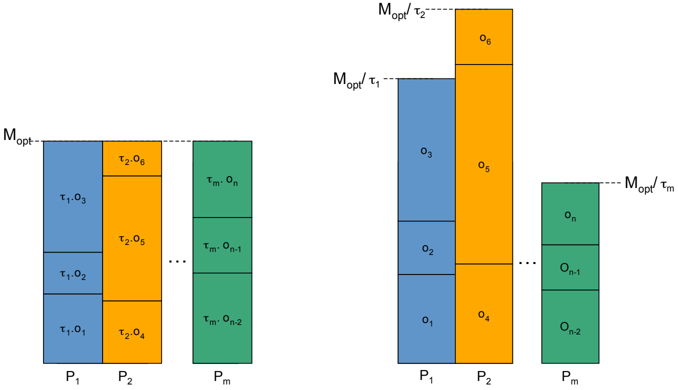
$E = \emptyset$

Makespan problem related to the 1-D bin-packing problem with variable bin size.



$E = \emptyset$

Makespan problem related to the 1-D bin-packing problem with variable bin size.



$$\sum_{j=1}^{m} \frac{M_{\text{opt}}}{\tau_i} = \sum_{i=1}^{n} o_i \Rightarrow M_{\text{opt}} = \frac{\sum_{i=1}^{n} o_i}{\sum_{j=1}^{m} \frac{1}{\tau_i}}$$

Independent tasks: the makespan problem

Gonzalez, Ibarra, Sahni 1977:

$$\left\{ \begin{array}{l} M_{\text{opt}} \geq \frac{\sum_{i=1}^{n} o_i}{\sum_{j=1}^{m} \frac{1}{\tau_i}} \\ n \geq m \end{array} \right. \Rightarrow \frac{M_{\text{LPT}}}{M_{\text{opt}}} \leq \frac{2m}{m+1} < 2$$

LPT: *Least Processing Time* scheduling heuristic.

Independent tasks: bound on the reliability

- $M_{\mathrm{LPT}} < 2 \cdot M_{\mathrm{opt}}$

Independent tasks: bound on the reliability

- $M_{\text{LPT}} < 2 \cdot M_{\text{opt}}$
- $\tilde{p}_{\text{succ}} = e^{-\sum \lambda_i M_{\text{opt}}}$

Independent tasks: bound on the reliability

- $M_{\mathrm{LPT}} < 2 \cdot M_{\mathrm{opt}}$
- $\tilde{p}_{\mathrm{succ}} = e^{-\sum \lambda_i M_{\mathrm{opt}}}$
- $p_{\mathrm{succ}} \geq e^{-\sum \lambda_i M_{\mathrm{LPT}}} > e^{-2\sum \lambda_i M_{\mathrm{opt}}} > \tilde{p}^2_{\mathrm{succ}}$

Independent tasks: bound on the reliability

- $M_{\text{LPT}} < 2 \cdot M_{\text{opt}}$
- $\tilde{p}_{\text{succ}} = e^{-\sum \lambda_i M_{\text{opt}}}$
- $p_{\text{succ}} \geq e^{-\sum \lambda_i M_{\text{LPT}}} > e^{-2\sum \lambda_i M_{\text{opt}}} > \tilde{p}_{\text{succ}}^2$

$\Rightarrow$ $p_{\text{fail}} \leq 2 \cdot \tilde{p}_{\text{fail}}$

Can we do better?

- We have proven that LPT is (2,2)-approximation algorithm (for $n \geq m$).

Can we do better?

- We have proven that LPT is (2,2)-approximation algorithm (for $n \geq m$).
- Can we help the user in choosing a better trade-off?

Can we do better?

- We have proven that LPT is (2,2)-approximation algorithm (for $n \geq m$).
- Can we help the user in choosing a better trade-off?
- Idea: limit the number of usable processors.

Can we do better?

- We have proven that LPT is (2,2)-approximation algorithm (for $n \geq m$).
- Can we help the user in choosing a better trade-off?
- Idea: limit the number of usable processors.
- Which processors to choose?

Can we do better?

- We have proven that LPT is (2,2)-approximation algorithm (for $n \geq m$).
- Can we help the user in choosing a better trade-off?
- Idea: limit the number of usable processors.
- Which processors to choose?
- The ones with the smallest $\lambda\tau$.

Can we do better?

- We have proven that LPT is (2,2)-approximation algorithm (for $n \geq m$).
- Can we help the user in choosing a better trade-off?
- Idea: limit the number of usable processors.
- Which processors to choose?
- The ones with the smallest $\lambda\tau$.
- Why?

Makespan/reliability Trade-off

Recall: scheduling all the tasks on the processors i such that $i = \text{argmin}(\tau_i \lambda_i)$ leads to the best possible reliability.

Makespan/reliability Trade-off

Recall: scheduling all the tasks on the processors i such that $i = \text{argmin}(\tau_i \lambda_i)$ leads to the best possible reliability.

Generalization :

Proposition

The best possible reliability among all the schedule with makespan at most M is achieved when:

Recall: scheduling all the tasks on the processors i such that $i = \text{argmin}(\tau_i \lambda_i)$ leads to the best possible reliability.

Generalization :

Proposition

The best possible reliability among all the schedule with makespan at most M is achieved when:

1. *tasks are mapped to $\tilde{m}$ processors in increasing order of $\lambda_i \tau_i$,*
2. *the $\tilde{m} - 1$ first processors execute tasks up to the date M ($C_i = M$),*
3. *the $\tilde{m}$ processor executes the remaining tasks ($C_{\tilde{m}} \leq M$).*

Makespan/reliability Trade-off

Recall: scheduling all the tasks on the processors i such that $i = \text{argmin}(\tau_i \lambda_i)$ leads to the best possible reliability.

Generalization :

Proposition

The best possible reliability among all the schedule with makespan at most M is achieved when:

1. *tasks are mapped to $\tilde{m}$ processors in increasing order of $\lambda_i \tau_i$,*
2. *the $\tilde{m} - 1$ first processors execute tasks up to the date M ($C_i = M$),*
3. *the $\tilde{m}$ processor executes the remaining tasks ($C_{\tilde{m}} \leq M$).*

Remark: such a schedule is not always feasible (it just gives a lower bound).

Outline

Generalizing Scheduling Heuristics: the HEFT case

- HEFT (*Heterogenous Earlisest Finish Time*) to RHEFT (*Reliable Heterogeneous Earlisest Finish Time)*.

Generalizing Scheduling Heuristics: the HEFT case

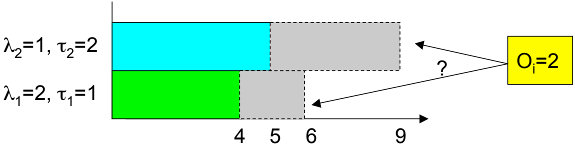
- HEFT (*Heterogenous Earlisest Finish Time*) to RHEFT (*Reliable Heterogeneous Earlisest Finish Time)*.
- HEFT: at each step of the heuristic map the task that finishes the earliest to this processors.

Generalizing Scheduling Heuristics: the HEFT case

- HEFT (*Heterogenous Earlisest Finish Time*) to RHEFT (*Reliable Heterogeneous Earlisest Finish Time)*.
- HEFT: at each step of the heuristic map the task that finishes the earliest to this processors.
- RHEFT: select the task that $T_{\mathrm{end}j} \times \lambda_j$ is minimum.

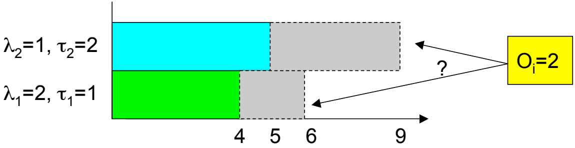
Generalizing Scheduling Heuristics: the HEFT case

- HEFT (*Heterogenous Earlisest Finish Time*) to RHEFT (*Reliable Heterogeneous Earlisest Finish Time)*.
- HEFT: at each step of the heuristic map the task that finishes the earliest to this processors.
- RHEFT: select the task that $T_{\mathrm{end}j} \times \lambda_j$ is minimum.



Generalizing Scheduling Heuristics: the HEFT case

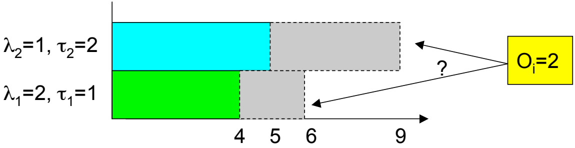
- HEFT (*Heterogenous Earlisest Finish Time*) to RHEFT (*Reliable Heterogeneous Earlisest Finish Time*).
- HEFT: at each step of the heuristic map the task that finishes the earliest to this processors.
- RHEFT: select the task that $T_{\text{end}j} \times \lambda_j$ is minimum.



- $T_{\text{end}1} = 6,\ T_{\text{end}1} \times \lambda_1 = 12$

Generalizing Scheduling Heuristics: the HEFT case

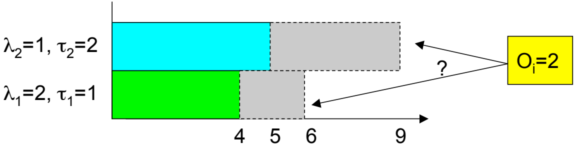
- HEFT (*Heterogenous Earlisest Finish Time*) to RHEFT (*Reliable Heterogeneous Earlisest Finish Time*).
- HEFT: at each step of the heuristic map the task that finishes the earliest to this processors.
- RHEFT: select the task that $T_{\mathrm{end}\,j} \times \lambda_j$ is minimum.



- $T_{\mathrm{end}\,1} = 6,\ T_{\mathrm{end}\,1} \times \lambda_1 = 12$
- $T_{\mathrm{end}\,2} = 9,\ T_{\mathrm{end}\,2} \times \lambda_2 = 9$

Generalizing Scheduling Heuristics: the HEFT case

- HEFT (*Heterogenous Earlisest Finish Time*) to RHEFT (*Reliable Heterogeneous Earlisest Finish Time)*.
- HEFT: at each step of the heuristic map the task that finishes the earliest to this processors.
- RHEFT: select the task that $T_{\mathrm{end}\,j} \times \lambda_j$ is minimum.



- $T_{\mathrm{end}\,1} = 6,\ T_{\mathrm{end}\,1} \times \lambda_1 = 12$
- $T_{\mathrm{end}\,2} = 9,\ T_{\mathrm{end}\,2} \times \lambda_2 = 9$

Easy to extend to other heuristics (sufferage, etc.).

Reliability/Makespan Trade-off

Two ways top find a good trade-off:

Reliability/Makespan Trade-off

Two ways top find a good trade-off:

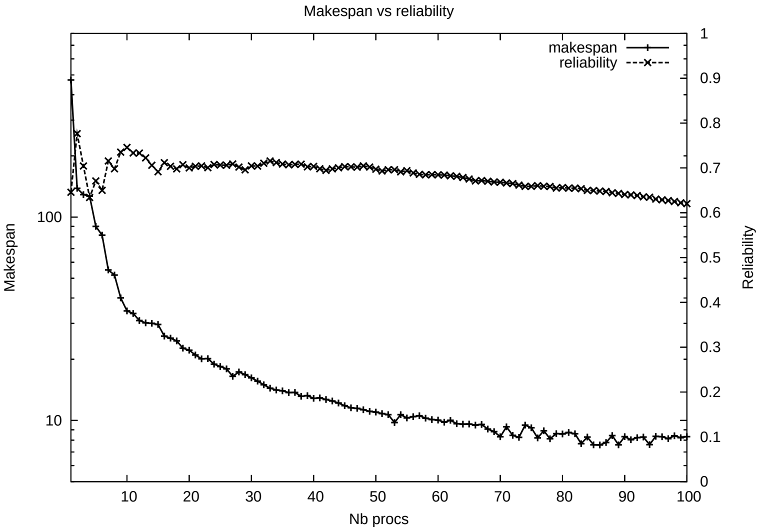
1. Choose a subset of processors; Q: which order?

Reliability/Makespan Trade-off

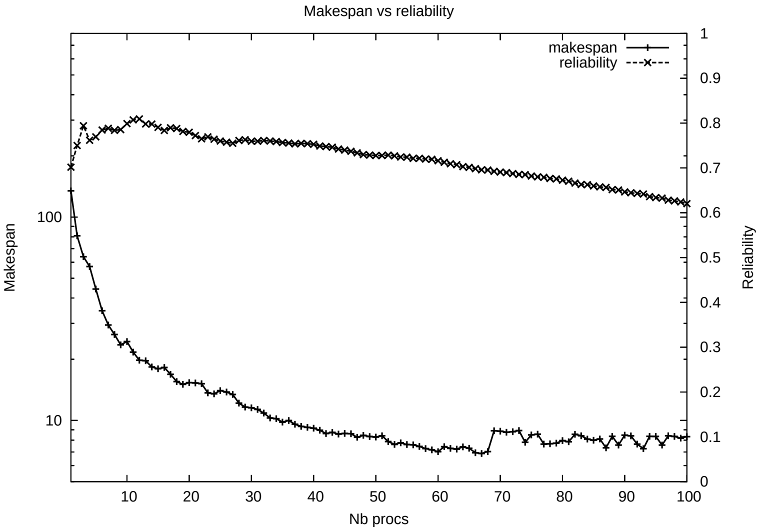
Two ways top find a good trade-off:

1. Choose a subset of processors; Q: which order?
2. Use a trade-off variable α ($\alpha = 1$ switch to HEFT, $\alpha = 0$ switch to RHEFT).

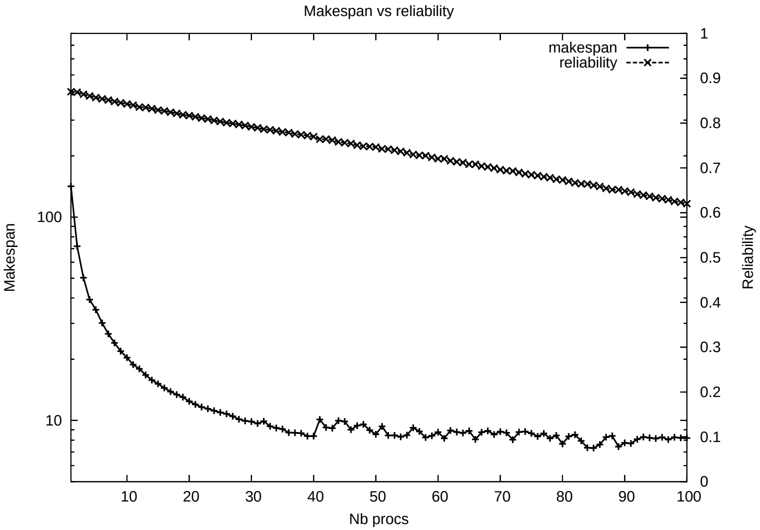
Ordering the processors: most reliable first



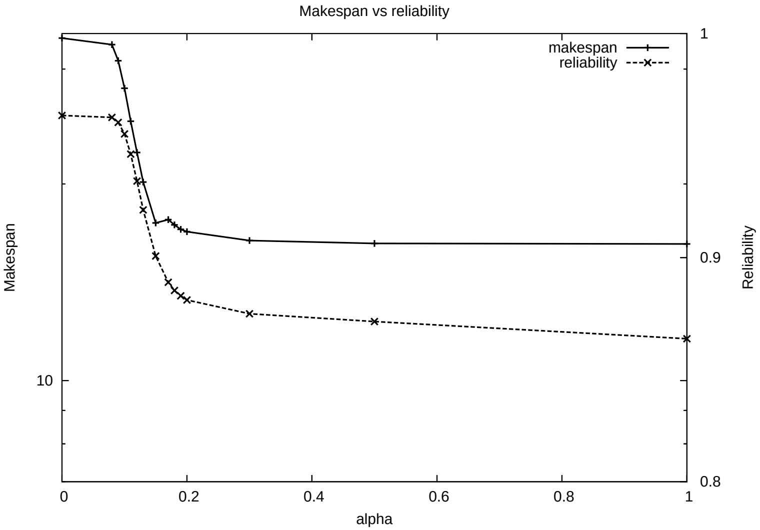
Ordering the processors: fastest first



Ordering the processors: smallest $\lambda\tau$ first



Trade-off variable



Outline

Conclusion

We have studied the problem of scheduling DAGs, with 2 objectives:

Conclusion

We have studied the problem of scheduling DAGs, with 2 objectives:

1. minimize makespan

Conclusion

We have studied the problem of scheduling DAGs, with 2 objectives:

1. minimize makespan
2. maximize reliability

Conclusion

We have studied the problem of scheduling DAGs, with 2 objectives:

1. minimize makespan
2. maximize reliability

Contribution:

Conclusion

We have studied the problem of scheduling DAGs, with 2 objectives:

1. minimize makespan
2. maximize reliability

Contribution:

- optimal algorithms for unitary independent tasks,

Conclusion

We have studied the problem of scheduling DAGs, with 2 objectives:

1. minimize makespan
2. maximize reliability

Contribution:

- optimal algorithms for unitary independent tasks,
- approximation algorithm for independent tasks ($n \geq m$),

Conclusion

We have studied the problem of scheduling DAGs, with 2 objectives:

1. minimize makespan
2. maximize reliability

Contribution:

- optimal algorithms for unitary independent tasks,
- approximation algorithm for independent tasks ($n \geq m$),
- simple way to generalize heuristics to this context,

Conclusion

We have studied the problem of scheduling DAGs, with 2 objectives:

1. minimize makespan
2. maximize reliability

Contribution:

- optimal algorithms for unitary independent tasks,
- approximation algorithm for independent tasks ($n \geq m$),
- simple way to generalize heuristics to this context,
- characterization of the role of the $\lambda\tau$ value.