Multi-users, multi-organizations, multi-objectives : a single approach

Denis Trystram (Grenoble University and INRIA)

Collection of results of 3 papers with:
Pierre-François Dutot (Grenoble University)
Krzysztof Rzadca (Polish-Japanese computing school, Warsaw)
Fanny Pascual (LIP6, Paris)
Erik Saule (Grenoble university)

Aussois, may 19, 2008
Goal

The evolution of high-performance execution platforms leads to physical or logical distributed entities (organizations) which have their own « local » rules, each organization is composed of multiple users that compete for the resources, and they aim to optimize their own objectives...

Construct a framework for studying such problems.

Work partially supported by the Coregrid Network of Excellence of the EC.
content

Brief review of basic (computational) models
Multi-users scheduling (1 resource)
Multi-users scheduling (m resources)
Multi-organizations scheduling (1 objective)
Multi-organizations with mixed objectives
Computational model

A set of users have some (parallel) applications to execute on a (parallel) machine.

The « machine » belongs or not to multiple organizations.

The objectives of the users are not always the same.
Multi-users optimization

Let us start by a simple case: several users compete for resources belonging to the same organization.

System centered problems (Cmax, load-balancing)
Users centered (minsum, maxstretch, flowtime)

**Motivation:**
Take the diversity of users’ wishes/needs into account
A simple example

Blue (4 tasks duration 3, 4, 4 and 5) has a program to compile ($C_{\text{max}}$)
Red (3 tasks duration 1, 3 and 6) is running experiments ($\sum Ci$)
m = 3 (machines)

Global LPT schedule
$C_{\text{max}} = 9$
$\sum Ci = 6 + 8 + 9 = 23$
A simple example

Blue (4 tasks duration 3,4,4 and 5) has a program to compile (Cmax)
Red (3 tasks duration 1,3 and 6) is running experiments ($\sum Ci$)
m=3 (machines)

Global LPT schedule
Cmax = 9
$\sum Ci = 6+8+9 = 23$

SPT schedule for red
Cmax = 8
$\sum Ci = 1+3+11 = 15$
Description of the problem

**Instance:** k users, user u submit n(u) tasks, processing time of task i belonging to u: \( p_i(u) \)
Completion time: \( C_i(u) \)
Each user can choose his-her objective among: \( C_{\text{max}}(u) = \max (C_i(u)) \) or \( \sum C_i(u) \) weighted or not

Multi-user scheduling problem:
MUSP(k':\(\sum C_i\);k'':\(C_{\text{max}}\)) where k'+k''=k
Complexity

[Agnitis et al. 2004], case m=1
MUSP(2:ΣC<sub>i</sub>) is NP-hard in the ordinary sense
MUSP(2:C<sub>max</sub>) and MUSP(1:ΣC<sub>i</sub>;1:C<sub>max</sub>) are polynomial

Thus, on m machines, all variants of this problem are NP-hard
We are looking for approximation (multi-objective)
MUSP\((k:C_{\text{max}})\)

**Inapproximability:**
no algorithm better than \((1,2,...,k)\)

**Proof:** consider the instance where each user has one unit task \(p_i(u)=1\) on one machine \(m=1\). \(C_{\text{max}}^*(u) = 1\) and there is no other choice than:

```
[ ] [ ] [ ] [ ] ...
```
MUSP(k: $C_{\text{max}}$)

**Inapproximability:**
no algorithm better than $(1, 2, \ldots, k)$

**Proof:** consider the instance where each user has one unit task ($p_i(u)=1$) on one machine. $C_{\text{max}}^*(u) = 1$ and there is no other choice than:

```
[ ] [ ] [ ] [ ]
```

Thus, there exists a user $u$ whose $C_{\text{max}}(u) = k$
MUSP\((k:C_{\text{max}})\)

Algorithm (multiCmax):
Given a \(\rho\)-approximation schedule \(\sigma\) for each user
\[ C_{\text{max}}(u) \leq \rho C_{\text{max}}^*(u) \]
Sort the users by increasing values of \(C_{\text{max}}(u)\)

Analysis:
multiCmax is a \((\rho, 2\rho, ..., k\rho)\)-approximation.
**MUSP**$(k: \Sigma C_i)$

**Inapproximability:** no algorithm better than $((k+1)/2, (k+2)/2, \ldots, k)$

**Proof:** consider the instance where each user has $x$ Tasks $p_i(u) = 2^{i-1}$.

Optimal schedule: $\Sigma C_i^* = 2^{x+1} - (x+2)$

SPT is Pareto Optimal (3 users blue, green and red):

![Diagram of user tasks]

For all $u$, $\Sigma C_i^{SPT}(u) = k(2^x -(x+1)) + (2^x -1) u$

Ratio to the optimal = $(k+u)/2$ for large $x$
MUSP\((k: \Sigma C_i)\)

Algorithm (single machine) Aggreg:
Let \( \sigma(u) \) be the schedule for user \( u \).
Construct a schedule by increasing order of \( \Sigma C_i(\sigma(u)) \)
(global SPT)
MUSP(k:ΣCi)

Algorithm (single machine) Aggreg:
Let \( \sigma(u) \) be the schedule for user u.
Construct a schedule by increasing order of \( \Sigma C_i(\sigma(u)) \)
(global SPT)
Algorithm (single machine) Aggreg:
Let $\sigma(u)$ be the schedule for user $u$.
Construct a schedule by increasing order of $\Sigma C_i(\sigma(u))$ (global SPT)
MUSP(k:ΣC_i)

Algorithm (single machine) Aggreg:
Let \( \sigma(u) \) be the schedule for user \( u \).
Construct a schedule by increasing order of \( \Sigma C_i(\sigma(u)) \)
(global SPT)
**MUSP(k:ΣC_i)**

Algorithm (single machine) Aggreg:
Let $\sigma(u)$ be the schedule for user $u$.
Construct a schedule by increasing order of $\Sigma C_i(\sigma(u))$
(global SPT)
MUSP($k: \sum C_i$)

Algorithm (single machine) Aggreg:
Let $\sigma(u)$ be the schedule for user $u$.
Construct a schedule by increasing order of $\sum C_i(\sigma(u))$
(global SPT)
Algorithm (single machine) Aggreg:
Let $\sigma(u)$ be the schedule for user $u$.
Construct a schedule by increasing order of $\Sigma C_i(\sigma(u))$ (global SPT)
Algorithm (single machine) Aggreg:
Let $\sigma(u)$ be the schedule for user $u$.
Construct a schedule by increasing order of $\Sigma C_i(\sigma(u))$ (global SPT)
MUSP\((k: \Sigma C_i)\)

Algorithm (single machine) Aggreg:
Let \( \sigma(u) \) be the schedule for user \( u \).
Construct a schedule by increasing order of \( \Sigma C_i(\sigma(u)) \)
(global SPT)
Algorithm (single machine) Aggreg:
Let $\sigma(u)$ be the schedule for user $u$.
Construct a schedule by increasing order of $\Sigma C_i(\sigma(u))$ (global SPT)
MUSP($k; \Sigma C_i$)

Algorithm (single machine) Aggreg:
Let $\sigma(u)$ be the schedule for user $u$.
Construct a schedule by increasing order of $\Sigma C_i(\sigma(u))$
(global SPT)

Analysis: Aggreg is $(k,k,...,k)$-approximation
MUSP\( (k:\Sigma C_i) \)

Algorithm (extension to \( m \) machines):
The previous property still holds on each machine
(using SPT individually on each machine)

Local SPT

Merge on each machine
MUSP(k: \sum C_i)

Algorithm (extension to m machines):
The previous property still holds on each machine
(using SPT individually on each machine)

Analysis:
we obtain the same bound as before.
Mixed case

\[ \text{MUSP}(k':\Sigma C_i;(k-k'):C_{\text{max}}) \]

A similar analysis can be done, see the paper with Erik Saule for more details
Complicating the model: Multi-organizations
Context: computational grids

Collection of independent clusters managed locally by an « organization ».
Preliminary:

Single resource: cluster

Independent applications are submitted locally on a cluster. They are represented by a precedence task graph.

An application is viewed as a usual sequential task or as a parallel rigid job (see [Feitelson and Rudolph] for more details and classification).
Local queue of submitted jobs

\[ \cdots \quad J3 \quad J2 \quad J1 \]
Job
Rigid jobs: the number of processors is fixed.
Useful definitions:
high jobs (those which require more than \( m/2 \) processors)
low jobs (the others).
Scheduling rigid jobs: Packing algorithms (batch)

Scheduling independent rigid jobs may be solved as a 2D packing Problem (strip packing).
Multi-organizations

n organizations.

Organization $k$

$\begin{array}{c}
\ldots \\
J3 \\
J2 \\
J1 \\
\end{array}$

$\begin{array}{c}
\ldots \\
\text{Cluster} \\
\text{m}_k \text{ processors} \\
\end{array}$
users submit their jobs locally
The organizations can cooperate
Constraints

$C_{\text{max}}(O_k)$: maximum finishing time of jobs belonging to $O_k$. Each organization aims at minimizing its own makespan.
Problem statement

MOSP: minimization of the « global » makespan under the constraint that no local schedule is increased.

**Consequence:** taking the restricted instance n=1 (one organization) and m=2 with sequential jobs, the problem is the classical 2 machines problem which is NP-hard. Thus, MOSP is NP-hard.
Multi-organizations

Motivation:
A non-cooperative solution is that all the organizations compute their local jobs (« my job first » policy).
However, such a solution is arbitrarily far from the global optimal (it grows to infinity with the number of organizations $n$).
See next example with $n=3$ for jobs of unit length.

\[\begin{array}{ccc}
O_1 & & \\
O_2 & & \\
O_3 & & \\
\end{array}\begin{array}{ccc}
O_1 & & \\
O_2 & & \\
O_3 & & \\
\end{array}\]

no cooperation

\[\begin{array}{ccc}
O_1 & & \\
O_2 & & \\
O_3 & & \\
\end{array}\begin{array}{ccc}
O_1 & & \\
O_2 & & \\
O_3 & & \\
\end{array}\]

with cooperation (optimal)
More sophisticated algorithms than the simple load balancing are possible: matching certain types of jobs may lead to bilaterally profitable solutions. However, it is a hard combinatorial problem.
Preliminary results

- List-scheduling: (2-1/m) approximation ratio for the variant with resource constraint [Garey-Graham 1975].
- HF: Highest First schedules (sort the jobs by decreasing number of required processors). Same theoretical guaranty but perform better from the practical point of view.
Analysis of HF (single cluster)

**Proposition.** All HF schedules have the same structure which consists in two consecutive zones of high (I) and low (II) utilization.

**Proof.** (2 steps)
By contradiction, no high job appears after zone (II) starts.
If we can not worsen any local makespan, the global optimum can not be reached.
If we can not worsen any local makespan, the global optimum can not be reached.

Best solution that does not increase $C_{\text{max}}(O_1)$
If we can not worsen any local makespan, the global optimum can not be reached.

Lower bound on approximation ratio greater than $\frac{3}{2}$.

best solution that does not increase $C_{\text{max}}(O_1)$
Using Game Theory?

We propose here a standard approach using Combinatorial Optimization.

Cooperative Game Theory may also be useful, but it assumes that players (organizations) can communicate and form coalitions. The members of the coalitions split the sum of their playoff after the end of the game. We assume here a centralized mechanism and no communication between organizations.
Multi-Organization Load-Balancing

1. Each cluster is running local jobs with Highest First
   \[ LB = \max (p_{\text{max}}, W/nm) \]
2. Unschedule all jobs that finish after 3LB.
3. Divide them into 2 sets (Ljobs and Hjobs)
4. Sort each set according to the Highest first order
5. Schedule the jobs of Hjobs backwards from 3LB
   on all possible clusters
6. Then, fill the gaps with Ljobs in a greedy manner
let consider a cluster whose last job finishes before 3LB
Feasibility (insight)

Zone (I)  Zone (II)  Zone (I)

3LB
Sketch of analysis

Proof by contradiction:
let us assume that it is not feasible,
and call $x$ the first job that does not fit in a cluster.

Case 1: $x$ is a small job.
Global surface argument

Case 2: $x$ is a high job.
Much more complicated,
see the paper for technical details
Proposition:
1. The previous algorithm is a 3-approximation (by construction)
2. The bound is tight (asymptotically)

Consider the following instance:
m clusters, each with 2m-1 processors
The first organization has m short jobs requiring each the full machine (duration $\varepsilon$) plus m jobs of unit length requiring m processors
All the m-1 others own m sequential jobs of unit length
Local HF schedules
Optimal (global) schedule: $C_{\text{max}}^* = 1 + \epsilon$
Multi-organization load-balancing: Cmax=3
We add an extra load-balancing procedure
Some experiments

- ▲ local
- ○ REMOLBA
- ◊ ILBA

![Graph showing the relationship between the number of organizations and some metric, with data points and error bars for different categories.](image)
Link with Game Theory?

We propose an approach based on combinatorial optimization

Can we use Game theory?

*players*: organizations or users

*objective*: makespan, minsum, mixed

Cooperative game theory: assume that players communicate and form coalitions.

Non cooperative game theory: key concept is Nash equilibrium which is the situation where the players donot have interest to change their strategy...

*stratégie*: collaborer ou non; *obj. global*: min makespan

Price of stability: best Nash equilibrium over the opt. solution
Conclusion

Single unified approach based on multi-objective optimization for taking into account the users’ need or wishes.

MOSP - good guaranty for $C_{\text{max}}, \Sigma C_i$ and mixed case remains to be studied
MUSP - « bad » guaranty but we can not obtain better with mow cost algorithms
Thanks for attention
Do you have any questions?