Minimizing the stretch when scheduling flows of divisible requests

Arnaud Legrand[1] Alan Su[2] Frédéric Vivien[2,3]

[1]Laboratoire ID-IMAG, Grenoble, France

[2]LIP, École normale supérieure de Lyon, France

[3]INRIA

November 24, 2005

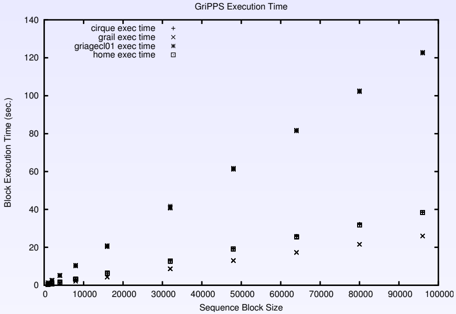
Outline

Problem context

- A distributed heterogeneous platform (cluster of clusters, grids, etc.).
- A collection of protein sequence databases:
  - text files ranging in size from several megabytes to several gigabytes
  - may be replicated across any number of nodes in the computational platform
  - *not* necessarily available to every computational node
- A workload composed of requests:
  - comparisons of regular-expression patterns against sequences in a given database
  - each request is independent from the others
  - request size varies depending on complexity of the patterns

Prototype application: GriPPS from l'Institut de Biologie et Chimie des Protéines

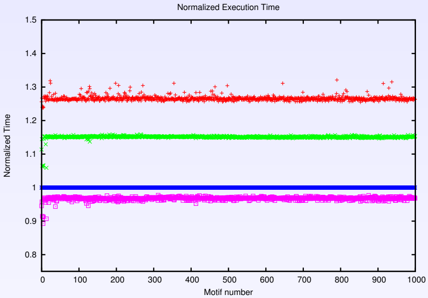
Application analysis : divisible loads



Preliminary analyses:

- benchmarks: variable size database partitions and a fixed motif set
- each benchmark run 10 times
- results justify a divisible workload model
- compact motif representation $\Rightarrow$ communication times are negligible compared to computation times

Application analysis : uniform computation model



- Determining relative processor speed
  - individual motif comparisons on reference computational resources
  - average over 40 iterations
  - normalize average execution times against a reference machine
- Task execution time estimated by
  - benchmark execution time
  - relative processor speed

Definitions

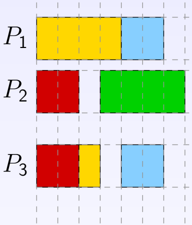
- Jobs J_1, ..., J_n
  - Job J_j arrives in the system at time r_j.
  - Job J_j has a size p_j.
- Machines M_1, ..., M_m
  - Machine M_i takes a time $c_{i,j}$ to process the job J_j.
  - $c_{i,j}$ is infinite if the job J_j needs a database that is not available on the machine M_i.
- Completion date C_1, ..., C_n
- Flow of job J_j : $F_j = C_j - r_j$ (time spent in the system)

Divisibility

- Each job is intrinsically divisible: at any given time different processors can compare a given pattern against different parts of a same database.

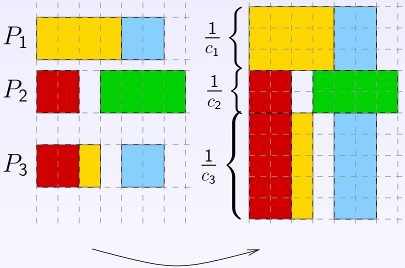
From divisible loads to the uni-processor case

Geometrical transformation of a divisible uniform problem into a preemptive uni-processor problem.



From divisible loads to the uni-processor case

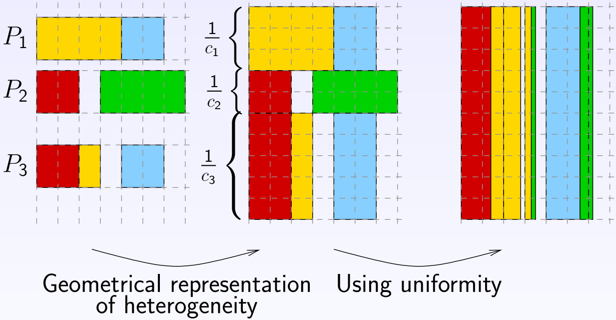
Geometrical transformation of a divisible uniform problem into a preemptive uni-processor problem.



Geometrical representation of heterogeneity

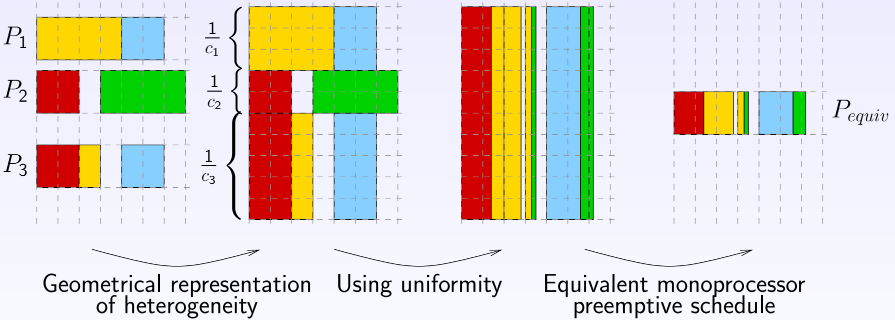
From divisible loads to the uni-processor case

Geometrical transformation of a divisible uniform problem into a preemptive uni-processor problem.



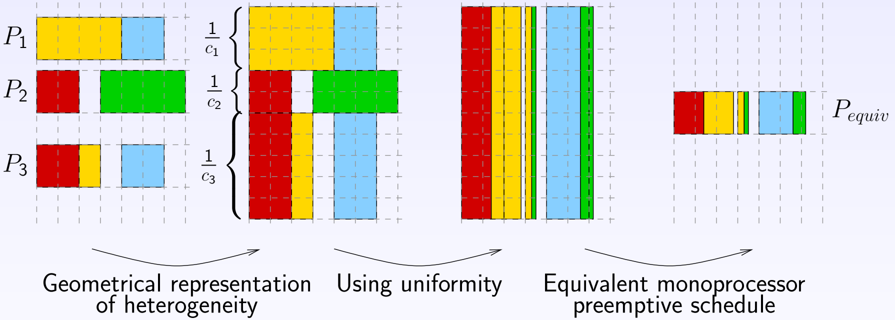
From divisible loads to the uni-processor case

Geometrical transformation of a divisible uniform problem into a preemptive uni-processor problem.



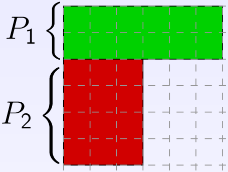
From divisible loads to the uni-processor case

Geometrical transformation of a divisible uniform problem into a preemptive uni-processor problem.



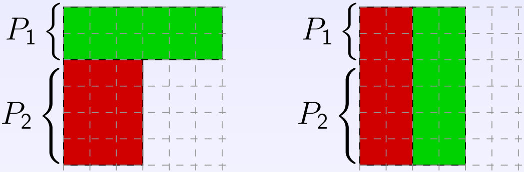
We only need to consider the uni-processor case...
Except that we are under the uniform case with availabilities.

From the uni-processor case to divisible loads



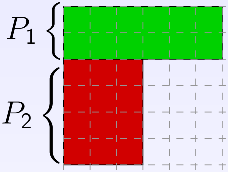
A: initial schedule

From the uni-processor case to divisible loads

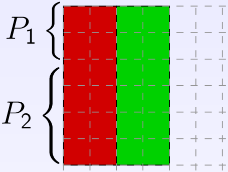


A: initial schedule B: uniform processing

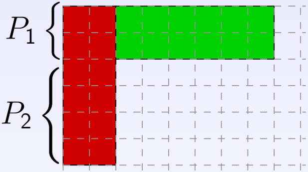
From the uni-processor case to divisible loads



A: initial schedule

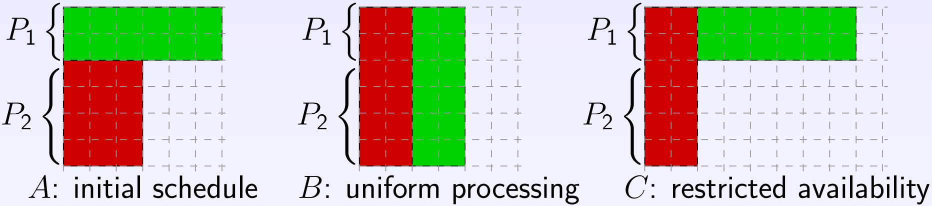


B: uniform processing



C: restricted availability

From the uni-processor case to divisible loads



Simple rule to extend schedules deisgned for the uni-processor case:

1: **while** some processors are idle **do**
2: Select the job with the highest priority and distribute its processing on all appropriate processors that are available.

Choosing a fair objective function

- Makespan: $\max_j C_j$.
  Optimization of the machines utilization.
  Release dates not taken into account.

Choosing a fair objective function

- Makespan: $\max_j C_j$.
  Optimization of the machines utilization.
  Release dates not taken into account.
- Average flow or sum-flow or average response time: $\sum_j (C_j - r_j)$.
  Optimization from the user point-of-view.

Choosing a fair objective function

- Makespan: $\max_j C_j$.
  Optimization of the machines utilization.
  Release dates not taken into account.
- Average flow or sum-flow or average response time: $\sum_j (C_j - r_j)$.
  Optimization from the user point-of-view.
  Drawback: can lead to starvation.

Choosing a fair objective function

- Makespan: $\max_j C_j$.
  Optimization of the machines utilization.
  Release dates not taken into account.
- Average flow or sum-flow or average response time: $\sum_j (C_j - r_j)$.
  Optimization from the user point-of-view.
  Drawback: can lead to starvation.
- Maximum flow or maximum response time: $\max_j (C_j - r_j)$.
  No starvation. Advantage for the long jobs. Worst case optimization.

Choosing a fair objective function

- Makespan: $\max_j C_j$.
  Optimization of the machines utilization.
  Release dates not taken into account.
- Average flow or sum-flow or average response time: $\sum_j (C_j - r_j)$.
  Optimization from the user point-of-view.
  Drawback: can lead to starvation.
- Maximum flow or maximum response time: $\max_j (C_j - r_j)$.
  No starvation. Advantage for the long jobs. Worst case optimization.
- Maximum weighted flow : $\max_j w_j (C_j - r_j)$.
  Enables us to give more importance to short jobs.

Choosing a fair objective function

- Makespan: $\max_j C_j$.
  Optimization of the machines utilization.
  Release dates not taken into account.

- Average flow or sum-flow or average response time: $\sum_j (C_j - r_j)$.
  Optimization from the user point-of-view.
  Drawback: can lead to starvation.

- Maximum flow or maximum response time: $\max_j (C_j - r_j)$.
  No starvation. Advantage for the long jobs. Worst case optimization.

- Maximum weighted flow : $\max_j w_j (C_j - r_j)$.
  Enables us to give more importance to short jobs.
  Special case: the *stretch*: $w_j = 1/\text{job size}$.

Minimizing the stretch

We focus on maximal and sum (average) stretch minimization.

Minimizing the stretch

We focus on maximal and sum (average) stretch minimization.

Theorem

Δ: ratio of the sizes of the largest and shortest jobs.
Consider any on-line algorithm of competitive ratio $\rho(\Delta) < \Delta$ for the sum-stretch minimization.

Then, there exists for this algorithm a sequence of jobs leading to starvation and for which the obtained max-stretch is arbitrarily greater than the optimal max-stretch.

Outline

Minimizing max- and sum-flow

- The max-flow is minimized by the *first come, first serve* rule.

- The sum-flow is minimized by the *shortest remaining processing time* (SRPT) heuristic.

Minimizing the sum-stretch

- Complexity of off-line problem: open.

Minimizing the sum-stretch

- Complexity of off-line problem: open.
- Polynomial Time Approximation Schemes (PTAS).

Minimizing the sum-stretch

- Complexity of off-line problem: open.
- Polynomial Time Approximation Schemes (PTAS).
- The competitive ratio of any online algorithm is at least 1.19485.

Minimizing the sum-stretch

- Complexity of off-line problem: open.
- Polynomial Time Approximation Schemes (PTAS).
- The competitive ratio of any online algorithm is at least 1.19485.
- Shortest Remaining Processing Time is 2-competitive.

Minimizing the sum-stretch

- Complexity of off-line problem: open.
- Polynomial Time Approximation Schemes (PTAS).
- The competitive ratio of any online algorithm is at least 1.19485.
- Shortest Remaining Processing Time is 2-competitive.
- Obvious extension : Shortest *Weighted* Remaining Processing Time. At any time t, SWRPT schedules the job J_j which minimizes $p_j \rho_t(j)$. SWRPT is *at best* 2-competitive.

Minimizing the sum-stretch

- Complexity of off-line problem: open.
- Polynomial Time Approximation Schemes (PTAS).
- The competitive ratio of any online algorithm is at least 1.19485.
- Shortest Remaining Processing Time is 2-competitive.
- Obvious extension : Shortest *Weighted* Remaining Processing Time. At any time t, SWRPT schedules the job J_j which minimizes $p_j \rho_t(j)$. SWRPT is *at best* 2-competitive.

The off-line case looks difficult
but simple approximation algorithms for the on-line framework.

Existence of a schedule of given max-stretch (1)

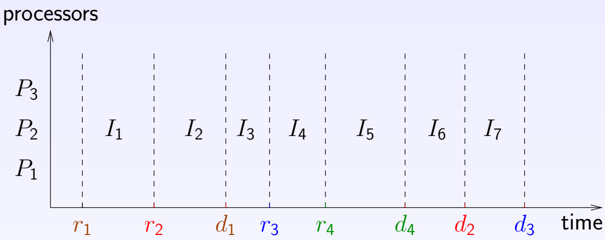
Existence of a schedule of max-stretch $\mathcal{S}$:

For each job J_j, $\frac{C_j - r_j}{p_j} \leqslant \mathcal{S}$

Equivalent to a deadline scheduling problem where $d_j(\mathcal{S}) = r_j + p_j \times \mathcal{S}$

Existence of a schedule of given max-stretch (2)

Set of all release dates and deadlines: $\{r_1, ..., r_n, d_1, ..., d_n\}$.



These dates, when sorted, define a set of n_{int} time intervals $I_1, ..., I_{n_{\text{int}}}$, with $1 \leqslant n_{\text{int}} \leqslant 2n - 1$.

$I_t = [\inf I_t, \sup I_t[$

$\alpha_{i,j}^{(t)}$: the fraction of job J_j processed by M_i during the interval I_t.

Existence of a schedule of given max-stretch (3)

1. *Release dates*:
$$\forall i, \forall j, \forall t, \quad r_j \geqslant \sup I_t(\mathcal{S}) \Rightarrow \alpha_{i,j}^{(t)} = 0$$
2. *Deadlines*:
$$\forall i, \forall j, \forall t, \quad d_j(\mathcal{S}) \leqslant \inf I_t(\mathcal{S}) \Rightarrow \alpha_{i,j}^{(t)} = 0$$
3. *Resources constraints*:
$$\forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.c_{i,j} \leqslant \sup I_t(\mathcal{S}) - \inf I_t(\mathcal{S})$$
4. *Job completion*:
$$\forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1$$

Deciding whether this system has a solution can be done in polynomial time.

Existence of a schedule of given max-stretch (3)

1. *Release dates*:

$$\forall i, \forall j, \forall t, \quad r_j \geqslant \sup I_t(\mathcal{S}) \Rightarrow \alpha_{i,j}^{(t)} = 0$$

2. *Deadlines*:

$$\forall i, \forall j, \forall t, \quad d_j(\mathcal{S}) \leqslant \inf I_t(\mathcal{S}) \Rightarrow \alpha_{i,j}^{(t)} = 0$$

3. *Resources constraints*:

$$\forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)} . c_{i,j} \leqslant \sup I_t(\mathcal{S}) - \inf I_t(\mathcal{S})$$

4. *Job completion*:

$$\forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1$$

Deciding whether this system has a solution can be done in polynomial time.

This system can be used to search for the best solution on a interval where the relative order of release dates and deadlines is constant.

Minimizing the max-stretch in the off-line case

- We compute the n_q special values of $\mathcal{S}$ for which one or more deadlines equals a release date or another deadline ($n_\mathsf{q} \leqslant n^2 - n$).
- Let $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{n_\mathsf{q}}$ be these special values of the objective, sorted.
  - by definition, no intersections of key dates: $\forall \mathcal{S}, \mathcal{S}_i \leqslant \mathcal{S} \leqslant \mathcal{S}_{i+1} \Rightarrow$ ordering of release dates and deadlines is unchanged
  - binary search on the set of special values of the objective, $\mathcal{S}_i$ (using the previously presented method with the objective interval $[\mathcal{S}_i, \mathcal{S}_{i+1}]$)
- The algorithm runs in polynomial time.

Minimizing the max-stretch in the online case (1)

Lower bound on algorithm competitivity:

Theorem

For three lengths of jobs, there is no $\frac{1}{2}\Delta^{\sqrt{2}-1}$*-competitive preemptive on-line algorithm minimizing max-stretch, where* Δ *is the ratio of the sizes of the largest and shortest jobs.*

Minimizing the max-stretch in the online case (2)

Two greedy approximation algorithms $\sqrt{\Delta}$-competitive:

Minimizing the max-stretch in the online case (2)

Two greedy approximation algorithms $\sqrt{\Delta}$-competitive:

1. Bender, Muthukrishnan, and Rajaraman (2002)
   For any job J_j, define a pseudo-stretch $\widehat{\mathcal{S}}_j(t)$:

$$\widehat{\mathcal{S}}_j(t) = \begin{cases} \frac{t-r_j}{\sqrt{\Delta}} & \text{if } 1 \leqslant p_j \leqslant \sqrt{\Delta}, \\ \frac{t-r_j}{\Delta} & \text{if } \sqrt{\Delta} < p_j \leqslant \Delta. \end{cases}$$

   Then, jobs are scheduled by decreasing pseudo-stretches.

Minimizing the max-stretch in the online case (2)

Two greedy approximation algorithms $\sqrt{\Delta}$-competitive:

1. Bender, Muthukrishnan, and Rajaraman (2002)
   For any job J_j, define a pseudo-stretch $\widehat{\mathcal{S}}_j(t)$:

$$\widehat{\mathcal{S}}_j(t) = \begin{cases} \frac{t-r_j}{\sqrt{\Delta}} & \text{if } 1 \leqslant p_j \leqslant \sqrt{\Delta}, \\ \frac{t-r_j}{\Delta} & \text{if } \sqrt{\Delta} < p_j \leqslant \Delta. \end{cases}$$

   Then, jobs are scheduled by decreasing pseudo-stretches.

2. Bender, Chahrabarti, and Muthukrishnan (1998).
   At each release date:
   - Computes the off-line max-stretch $\mathcal{S}$.
   - Schedule the jobs *earliest deadline first* with deadlines defined by $\sqrt{\Delta} \times \mathcal{S}$.

   Problem : only tries to optimize the most constraining jobs.

Minimizing the max-stretch in the online case (3)

1. Preempt the running job (if any).
2. Compute the best achievable max-stretch $\mathcal{S}$, considering the decisions already made.
3. With the deadlines and intervals defined by the max-stretch $\mathcal{S}$, solve:

$$
\text{MINIMIZE} \quad \sum_{j=1}^{n} \sum_{t} \left(\sum_{i=1}^{m} \alpha_{i,j}^{(t)} \right) \frac{\sup I_t(\mathcal{S}) + \inf I_t(\mathcal{S})}{2} \quad \text{, WHILE}
$$

$$
\begin{cases}
\text{(1a)} \quad \forall i, \forall j, \forall t, \quad r_j \geqslant \sup I_t(\mathcal{S}) \Rightarrow \alpha_{i,j}^{(t)} = 0 \\
\text{(1b)} \quad \forall i, \forall j, \forall t, \quad d_j(\mathcal{S}) \leqslant \inf I_t(\mathcal{S}) \Rightarrow \alpha_{i,j}^{(t)} = 0 \\
\text{(1c)} \quad \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.c_{i,j} \leqslant \sup I_t(\mathcal{S}) - \inf I_t(\mathcal{S}) \\
\text{(1d)} \quad \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1
\end{cases}
\qquad (1)
$$

(Heuristic approximation of a rational relaxation of the sum-stretch)

No guarantee !

Conclusion

Minimizing the sum-stretch

- Offline case: looks difficult.
- Online case : rather easy.

Minimizing the max-stretch

- Offline case: polynomial time.
- Online case : very difficult.

and in practice ?

Outline

Simulation settings

- **platforms** of 3, 10, and 20 homogeneous clusters with 10 processors each;
- **applications** with 3, 10, and 20 distinct reference databases;
- **database availabilities** of 30%, 60%, and 90% for each database;
- **workload density factors** of 0.75, 1.0, 1.25, 1.5, 2.0, and 3.0.

Simulation results

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE	1.0000	0.0003	1.0167	1.6729	0.3825	4.4468
ONLINE	1.0025	0.0127	2.0388	1.0806	0.0724	2.0343
ONLINE-EDF	1.0024	0.0127	2.0581	1.0775	0.0708	2.0392
ONLINE-EGDF	1.0781	0.1174	2.4053	1.0021	0.0040	1.0707
BENDER98 [1]	1.0798	0.1315	2.0978	1.0024	0.0044	1.0530
SWRPT	1.0845	0.1235	2.5307	1.0002	0.0012	1.0458
SRPT	1.0939	0.1299	2.3741	1.0044	0.0055	1.0907
SPT	1.1147	0.1603	2.8295	1.0027	0.0054	1.1195
BENDER02	3.4603	3.0260	28.4016	1.2053	0.2417	5.2022
MCT-DIV	6.3385	7.4375	73.4019	1.3732	0.5628	11.0440
MCT	27.0124	20.1083	129.6119	50.9840	36.9797	157.8909

Table: Aggregate statistics over all 162 platform/application configurations

Outline

Conclusion

Minimizing the sum-stretch

- Offline case: looks difficult.
- Online case : rather easy.

Minimizing the max-stretch

- Offline case: polynomial time.
- Online case : very difficult.

In practice

- SWRPT and Online-EDF very good
- ... but SWRPT may lead to starvation
- sum-stretch not enough discriminating ?