Scheduling and Data Redistribution Strategies on Star Platforms

Loris Marchal Veronika Rehn Yves Robert
Frédéric Vivien
Laboratoire de l'Informatique du Parallélisme
UMR CNRS-ENS Lyon-INRIA-UCBL 5668
Lyon, France
{Loris.Marchal, Veronika.Rehn, Yves.Robert, Frederic.Vivien}@ens-lyon.fr

Abstract

In this work we are interested in the problem of scheduling and redistributing data on master-slave platforms. We consider the case were the workers possess initial loads, some of which having to be redistributed in order to balance their completion times.

We assume that the data consists of independent and identical tasks. As the general case is NP-complete in the strong sense, we propose three heuristics. Simulations consolidate the theoretical results.

1. Introduction

In this work we consider the problem of scheduling and redistributing data on master-slave architectures in star topologies. Because of variations in the resource performance (CPU speed or communication bandwidth), or because of unbalanced amounts of current load on the workers, data must be redistributed between the participating processors, so that the updated load is better balanced in terms that the overall processing finishes earlier.

We adopt the following abstract view of our problem. There are $m + 1$ participating processors $P_0, P_1, \ldots, P_m$, where P_0 is the master. Each processor P_k, $1 \leq k \leq m$ initially holds L_k data items. During our scheduling process we try to determine which processor P_i should send some data to another worker P_j to equilibrate their finishing times. The goal is to minimize the global makespan, that is the time until each processor has finished to process its data. Furthermore we suppose that each communication link is fully bidirectional, with the same bandwidth for receptions and sendings. This assumption is quite realistic in practice, and does not change the complexity of the scheduling problem, which we prove NP-complete in the strong sense.

We assume that data items consist in independent and uniform (same-size) tasks. There are many practical applications who use fixed identical and independent tasks. A famous example is BOINC [2], the Berkeley Open Infrastructure for Network Computing, an open-source software platform for volunteer computing. It works as a centralized scheduler that distributes tasks for participating applications. These projects consists in the treatment of computation extensive and expensive scientific problems of multiple domains, such as biology, chemistry or mathematics. SETI@home [14] for example uses the accumulated computation power for the search of extraterrestrial intelligence. In the astrophysical domain, Einstein@home [5] searches for spinning neutron stars using data from the LIGO and GEO gravitational wave detectors. To get an idea of the task dimensions, in this project a task is about 12 MB and requires between 5 and 24 hours of dedicated computation. Also, from a theoretical viewpoint, the scheduling problem is obviously NP complete when tasks have different sizes (trivial reduction from 2-PARTITION [6], which provides yet another reason to restrict to same-size tasks.

As already mentioned, we suppose that all data are initially situated on the workers, which leads us to a kind of redistribution problem. Existing redistribution algorithms have a different objective. Neither do they care how the degree of imbalance is determined, nor do they inclu the computation phase in their optimizations. They expect that a load-balancing algorithm has already taken place. With help of these results, a redistribution algorithm determines the required communications and organizes them in minimal time. We could use such an approach: redistribute the data first, and then enter a purely computational phase. But our problem is more complicated as we suppose that communication and computation can overlap, i.e., every worker can start computing its initial data while the redistribution process takes place.

To summarize our problem: as the participating workers are not equally charged and/or because of different resource

15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing (PDP'07)
0-7695-2784-1/07 $20.00 © 2007 IEEE

performance, they might not finish their computation process at the same time. So we are looking for mechanisms on how to redistribute the loads in order to finish the global computation process in minimal time under the hypothesis that charged workers can compute at the same time as they communicate.

The rest of this paper is organized as follows. The problem framework and corresponding complexity results are detailed in Section 2. The presentation of three heuristics for heterogeneous platforms is the subject in Section 3. Simulation results are shown in Section 4. Section 5 briefly presents some related work. Finally, we give some conclusions in Section 6.

2. Framework

We consider a *star network* $S = P_0, P_1, \ldots, P_m$ shown in Figure 1. The processor P_0 is the master and the m remaining processors P_i, $1 \leq i \leq m$, are workers. The initial data are distributed on the workers, so every worker P_i possesses a number L_i of initial tasks. All tasks are independent and identical. As we assume a linear cost model, each worker P_i has a (relative) computing power w_i for the computation of one task: it takes $X.w_i$ time units to execute X tasks on the worker P_i. The master P_0 can communicate with each worker P_i via a communication link. A worker P_i can send some tasks via the master to another worker P_j to decrement its execution time. It takes $X.c_i$ time units to send X units of load from P_i to P_0 and $X.c_j$ time units to send these X units from P_0 to a worker P_j. Without loss of generality we assume that the master is not computing, and only communicating.

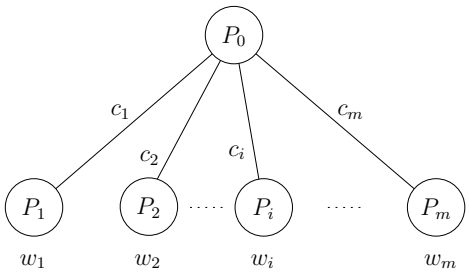


Figure 1. Example of a star network.

We use the bidirectional one-port model for communications. This means, that the master can only send data to, and receive data from, a single worker at a given time-step. But it can simultaneously receive a data and send one. A given worker cannot start an execution before it has terminated the reception of the message from the master; similarly, it cannot start sending the results back to the master before finishing the computation.

Comm.	Comp.	Difficulty
Hom.	Hom.	simple greedy algorithm
Hom.	Het.	complicated algorithm
Het.	Hom.	?
Het.	Het.	NP-strong

Table 1. Impact.

The objective function is to minimize the makespan, that is the time at which all loads have been processed. We look for a schedule σ that minimizes this objective. However, as shown in Table 1, the minimization problem is NP-complete (in the strong sense) for fully heterogeneous platforms [10]. Therefore, we propose three heuristics, with various complexities, and aim at assessing their performances.

3. Heuristics for heterogeneous platforms

3.1. Best Balance Algorithm - BBA

The first heuristic is a simple greedy algorithm BEST-BALANCE ALGORITHM (BBA) (see Algorithm 1). On heterogeneous platforms, at each step BBA optimizes the local makespan. The idea of BBA is the following: at each iteration, we look if we could finish earlier if we redistribute a task. If so, we schedule the task, if not, we stop redistributing. It turns out that this simple algorithm is optimal [10] for fully homogeneous processors, i.e., platforms with identical processors and communication link bandwidths.

Notations used in BBA: BBA schedules one task per iteration i. Let $L_k^{(i)}$ denote the number of tasks of worker k after iteration i, i.e., after i tasks were redistributed. The date at which the master has finished receiving the i-th task is denoted by $\mathsf{m\_in}^{(i)}$. In the same way we call $\mathsf{m\_out}^{(i)}$ the date at which the master has finished sending the i-th task. Let $\mathsf{end}_k^{(i)}$ be the date at which worker k would finish to process the load it would hold if exactly i tasks are redistributed. The worker k in iteration i with the biggest finish time $\mathsf{end}_k^{(i)}$, who is chosen to send one task in the next iteration, is called sender. We call receiver the worker k with smallest finish time $\mathsf{end}_k^{(i)}$ in iteration i who is chosen to receive one task in the next iteration.

In iteration $i = 0$ we are in the initial configuration: All workers own their initial tasks $L_k^{(0)} = L_k$ and the makespan of each worker k is the time it needs to compute all its tasks: $\mathsf{end}_k^{(0)} = L_k^{(0)} \times w$. $\mathsf{m\_in}^{(0)} = \mathsf{m\_out}^{(0)} = 0$.

Principle of BBA: In each iteration i, we compute the time $\mathsf{end}_k^{(i-1)}$ it would take worker k to process $L_k^{(i-1)}$ tasks. A worker with the biggest finish time $\mathsf{end}_k^{(i-1)}$ is

Algorithm 1: Best Balance Algorithm

$i \leftarrow 0$; $\mathsf{m\_in}^{(i)} \leftarrow 0$; $\mathsf{m\_out}^{(i)} \leftarrow 0$;
$\forall k L_k^{(0)} \leftarrow L_k$;
$\mathsf{end}_k^{(0)} \leftarrow L_k^{(0)} \times w_k$;
while *true* **do**
  $\mathsf{sender} \leftarrow \max_k \mathsf{end}_k^{(i)}$;
  $\mathsf{m\_in}^{(i+1)} \leftarrow \mathsf{m\_in}^{(i)} + c_{\mathsf{sender}}$;
  $\mathsf{task\_arr\_worker} =$
  $\max(\mathsf{m\_in}^{(i+1)}, \mathsf{m\_out}^{(i)}) + c_{\mathsf{sender}}$;
  foreach k **do**
    $\widetilde{\mathsf{end}}_k^{(i+1)} \leftarrow$
    $\max(\mathsf{end}_k^{(i+1)}, \mathsf{task\_arr\_worker}) + w_k$
  select receiver such that
  $\widetilde{\mathsf{end}}_{\mathsf{receiver}}^{(i+1)} = \min_k \widetilde{\mathsf{end}}_k^{(i+1)}$ and if there are several processors with the same minimum $\widetilde{\mathsf{end}}_k^{(i+1)}$, choose one with smallest $\mathsf{end}_k^{(i)}$;
  if $\mathit{end}_{\mathit{sender}}^{(i)} \leq \widetilde{\mathit{end}}_{\mathit{receiver}}^{(k+1)}$ **then**
    break; /* we cannot improve the makespan */
  else
    /* we improve the makespan by sending the task to the receiver */
    $\mathsf{m\_out}^{(i+1)} \leftarrow \mathsf{task\_arr\_worker}$;
    $\mathsf{end}_{\mathsf{sender}}^{(i+1)} \leftarrow \mathsf{end}_{\mathsf{sender}}^{(i)} - w_{\mathsf{sender}}$;
    $L_{\mathsf{sender}}^{(i+1)} \leftarrow L_{\mathsf{sender}}^{(i)} - 1$;
    $\mathsf{end}_{\mathsf{receiver}}^{(i+1)} \leftarrow \widetilde{\mathsf{end}}_{\mathsf{receiver}}^{(i+1)}$;
    $L_{\mathsf{receiver}}^{(i+1)} \leftarrow L_{\mathsf{receiver}}^{(i)} + 1$;
    foreach $j \neq \mathit{receiver}$ *and* $j \neq \mathit{sender}$ **do**
      $\mathsf{end}_j^{(i+1)} \leftarrow \mathsf{end}_j^{(i)}$; $L_j^{(i+1)} \leftarrow L_j^{(i)}$;
    $i \leftarrow i + 1$

arbitrarily chosen as sender, he is called sender. Then we compute the temporary finish times $\widetilde{\mathsf{end}}_k^{(i)}$ of each worker if it would receive from sender the i-th task. A worker with the smallest temporary finish time $\widetilde{\mathsf{end}}_k^{(i)}$ will be the receiver, called receiver. If there are multiple workers with the same temporary finish time $\widetilde{end}_k^{(i)}$, we take the worker with the smallest finish time $\mathsf{end}_k^{(i-1)}$. If the finish time of sender is strictly larger than the temporary finish time $\widetilde{\mathsf{end}}_{\mathsf{sender}}^{(i)}$ of sender, sender sends one task to receiver and iterate. Otherwise stop.

3.2. Moore Based Binary-Search Algorithm

Another heuristic is the utilization of a more complex algorithm MOORE BASED BINARY-SEARCH ALGORITHM (MBBSA). This algorithm is optimal [10] for platforms with homogeneous communication links and heterogeneous

Algorithm 2: Algorithm to optimize the makespan.

/* idea: make a binary search of
$M \in [\min(f_i), \max(f_i)]$ */
input: $w_i = \frac{\alpha_i}{\beta_i}, \alpha_i, \beta_i \in \mathbb{N} \times \mathbb{N}^*$,
$c_i = \frac{\gamma_i}{\delta_i}, \gamma_i, \delta_i \in \mathbb{N} \times \mathbb{N}^*$;
$\lambda \leftarrow \text{lcm}\{\beta_i, \delta_i\},\ 1 \leq i \leq m$
$precision \leftarrow \frac{1}{\lambda}$;
$lo \leftarrow \min(f_i)$; $hi \leftarrow \max(f_i)$;
procedure *binary-Search(lo, hi):*
$gap \leftarrow |lo - hi|$;
while $gap > precision$ **do**
  $M \leftarrow (lo + hi)/2$;
  $found \leftarrow$ MBBSA (M);
  if $\neg found$ **then**
    /* M is too small */
    $lo \leftarrow M$;
  else
    /* M is maybe too big */
    $hi \leftarrow M$;
    $\sigma \leftarrow$ found schedule;
  $gap \leftarrow |lo - hi|$;
return σ;

platforms. As the name says, this heuristic is based on MOORE'S ALGORITHM [3, 11], whose aim is to maximize the number of tasks to be processed in-time, i.e., before tasks exceed their deadlines. This algorithm gives a solution to the $1||\sum U_j$ problem when the maximum number, among n tasks, has to be processed in time on a single machine.

For a given makespan, we compute if there exists a possible schedule to finish all work in time. If there is one, we optimize the makespan by a binary search (cf. Algorithm 2).

Framework and notations for MBBSA: We keep the star network of Section 2. We suppose m heterogeneous workers who own initially a number L_i of identical independent tasks.

Let M denote the objective makespan for the searched schedule σ and f_i the time needed by worker i to process its initial load. During the algorithm execution we divide all workers in two subsets, where S is the set of senders ($s_i \in S$ if $f_i > M$) and R the set of receivers ($r_i \in R$ if $f_i < M$). As our algorithm is based on Moore's, we need a notation for deadlines. Let $d_{r_i}^{(k)}$ be the deadline to receive the k-th task on receiver r_i. l_{s_i} denotes the number of tasks sender i sends to the master and l_{r_i} stores the number of tasks receiver i is able to receive from the master. With help of these values we can determine the total amount of tasks that must be sent as $L_{send} = \sum_{s_i} l_{s_i}$. The total amount of task if all receivers receive the maximum amount of tasks they are able to receive is $L_{recv} = \sum_{r_i} l_{r_i}$. Finally, let L_{sched}

be the maximal amount of tasks that can be scheduled by the algorithm.

Principle of MBBSA: Considering the given makespan we determine overcharged workers, which can not finish all their tasks within this makespan. These overcharged workers will then send some tasks to undercharged workers, such that all of them can finish processing within the makespan. The algorithm solves the following two questions: Is there a possible schedule such that all workers can finish in the given makespan? In which order do we have to send and receive to obtain such a schedule? The pseudo code of MBBSA is described in Algorithm 3.

Phases of MBBSA:

Phase 1 decides which of the workers will be senders and which receivers, depending of the given makespan. Senders are workers which are not able to process all their initial tasks in time, whereas receivers are workers which could treat more tasks in the given makespan M than they hold initially. So sender P_i has a finish time $f_i > M$, i.e., the time needed to compute their initial tasks is larger than the given makespan M. Conversely, P_i is a receiver if it has a finish time $f_i < M$.

Phase 2 fixes how many transfers have to be scheduled from each sender such that the senders all finish their remaining tasks in time. Sender s_i will have to send an amount of tasks $l_{s_i} = \left\lceil \frac{f_{s_i} - T}{w_{s_i}} \right\rceil$.

Phase 3 computes for each receiver the deadline of each of the tasks it can receive, i.e., a pair $(d_{r_j}^{(i)}, r_j)$ that denotes the i-th deadline of receiver r_j. Beginning at the makespan M one measures when the last task has to arrive on the receiver such that it can be processed in time. So the latest moment that a task can arrive so that it can still be computed on receiver r_j is $T - w_{r_j}$, and so on.

Phase 4 is the proper scheduling step: The master decides which tasks have to be scheduled on which receivers and in which order. Starting at time $t = c$ (this is the time, when the first task arrives at the master), the master can start scheduling the tasks on the receivers. For this purpose the deadlines (d, r_j) are ordered by non-decreasing d-values. In the same manner as in Moore's algorithm, an optimal schedule σ is computed by adding one by one tasks to the schedule: considering the deadline (d, r_j), we add a task to processor r_j. If the communication takes too long and the deadline is not met, the last reception is suppressed from σ and we continue. If the schedule is able to send at least L_{send} tasks the algorithm succeeds, otherwise it fails.

Algorithm 3: Moore Based Binary-Search Algorithm

initialize f_i for all workers i, $f_i = L_i \times w_i$;
compute R and S, order S by non-decreasing values c_i such that $c_{s_1} \leq c_{s_2} \leq \ldots$;
foreach $s_i \in S$ **do**
  $l_{s_i} \leftarrow \left\lceil \frac{f_{s_i} - T}{w_{s_i}} \right\rceil$;
  if $\left\lfloor \frac{T}{c_{s_i}} \right\rfloor < l_{s_i}$ **then**
    return $(false, \emptyset)$; /* M too small */
total number of tasks to send: $L_{send} \leftarrow \sum_{s_i} l_{s_i}$;
$D \leftarrow \emptyset$;
foreach $r_i \in R$ **do**
  $l_{r_i} \leftarrow 0$;
  while $f_{r_i} \leq M - (l_{r_i} + 1) \times w_{r_i}$ **do**
    $l_{r_i} \leftarrow l_{r_i} + 1$;
    $d_{r_i}^{(l_{r_i})} \leftarrow M - (l_{r_i} \times w_{r_i})$;
    $D \leftarrow D \cup (d_{r_i}^{(l_{r_i})}, r_i)$;
# of tasks that can be received: $L_{recv} \leftarrow \sum_{r_i} l_{r_i}$;
senders send in non-decreasing order of values c_{s_i};
order deadline-list D by non-decreasing values of deadlines d_{r_i} and rename the deadlines in this order from 1 to L_{recv};
$\sigma \leftarrow \emptyset$; $t \leftarrow c_{s_1}$; $L_{sched} = 0$;
for $i = 1$ *to* L_{recv} **do**
  $(d_i, r_i) \leftarrow i$-th element $(d_{r_k}^{(j)}, r_k)$ of D;
  $\sigma \leftarrow \sigma \cup \{r_i\}$;
  $t \leftarrow t + c_{r_i}$; $L_{sched} \leftarrow L_{sched} + 1$;
  if $t > d_i$ **then**
    Find (d_j, r_j) in σ s.t. c_{r_j} value is largest;
    $\sigma \leftarrow \sigma \backslash \{(d_j, r_j)\}$;
    $t \leftarrow t - c_{r_j}$; $L_{sched} \leftarrow L_{sched} - 1$;
return $((L_{sched} \geq L_{send}), \sigma)$;

3.3. Reversed Binary-Search Algorithm

We propose a third heuristic: the REVERSED BINARY-SEARCH ALGORITHM (see Algorithm 4 for details). This algorithm copies the idea of the introduction of deadlines. Contrary to MBBSA this algorithm traverses the deadlines in reversed order, wherefrom the name. Starting at a given makespan, R-BSA schedules all tasks as late as possible until no more task can be scheduled.

R-BSA can be divided into four phases:

Phase 1 is the same as in MBBSA. It decides which of the workers will be senders and which receivers, depending of the given makespan.

Phase 2 fixes how many transfers have to be scheduled from each sender such that the senders all finish their

Algorithm 4: Reversed Binary-Search Algorithm

$T \leftarrow M$; $L_{sched} \leftarrow 0$; $\sigma \leftarrow \emptyset$;
$\forall k\ L_k^{(0)} \leftarrow L_k$;
initialize end_i for all workers i: $\mathsf{end}_i = L_i \times w_i$;
compute R and S, order S by non-decreasing values c_i: $c_{s_1} \leq c_{s_2} \leq \ldots$
$\mathsf{m\_in} \leftarrow c_{s_1}$;
foreach $s_i \in S$ **do**
  $l_{s_i} \leftarrow \left\lceil \frac{\mathsf{end}_{s_i} - T}{w_{s_i}} \right\rceil$;
  if $\left\lfloor \frac{T}{c_{s_i}} \right\rfloor < l_{s_i}$ **then**
    return $(false, \emptyset)$; /* M too small */
total number of tasks to sent: $L_{send} \leftarrow \sum_{s_i} l_{s_i}$;
$\forall r_i \in R$ **if** $end_{r_i} \leq T$ **then** $\mathsf{begin}_{r_i} \leftarrow T$;
while *true* **do**
  choose receiver such that $\max_i(\min(\mathsf{begin}_i - w_i, T)) - c_i$ is maximal and that the schedule is feasible: the task must fit in the idle gap of the worker: $(\mathsf{begin}_{\mathsf{receiver}} - w_{\mathsf{receiver}} \geq \mathsf{end}_{\mathsf{receiver}})$ and the task has to be arrived at the master: $(\mathsf{begin}_{\mathsf{receiver}} - w_{\mathsf{receiver}} - c_{\mathsf{receiver}} \geq \mathsf{m\_in})$;
  if *no receiver' found* **then**
    return $((L_{sched} \leq L_{send}), \sigma)$;
  $\mathsf{begin}_{\mathsf{receiver}} \leftarrow \mathsf{begin}_{\mathsf{receiver}} - w_{\mathsf{receiver}}$;
  $T \leftarrow \mathsf{begin}_{\mathsf{receiver}} - c_{\mathsf{receiver}}$;
  $L_{sched} \leftarrow L_{sched} + 1$;
  $\sigma \leftarrow \sigma \cup \{\mathsf{receiver}\}$;
  $i \leftarrow i + 1$;
return $((L_{sched} \geq L_{send}), \sigma)$;

remaining tasks in time. This phase is also identical to MBBSA.

Phase 3 computes for each receiver at which time it can start with the computation of the additional tasks, this is in general the given makespan.

Phase 4 again is the proper scheduling step: Beginning at the makespan we fill backward the idle times of the receiving workers. So the master decides which tasks have to be scheduled on which receivers and in which order. The master chooses a worker that can start to receive the task as late as possible and still finish it in time.

4. Simulations

In this section we present the results of our simulation experiences of the presented algorithms and heuristics on multiple platforms. We study the heuristics that we presented in Section 3.

4.1. Experimental plan

All simulations were made with SIMGRID [9, 16]. SimGrid is a toolkit that provides several functionalities for the simulation of distributed applications in heterogeneous distributed environments. The toolkit is distributed into several layers and offers several programming environments, such as XBT, the core toolbox of SimGrid or SMPI, a library to run MPI applications on top of a virtual environment. The access to the different components is ensured via Application Programming Interfaces (API). We use the module MSG to create our entities.

The simulations were made on automatically created random platforms of four types: We analyze the behavior on fully homogeneous and fully heterogeneous platforms and the mixture of both, i.e., platforms with homogeneous communication links and heterogeneous workers and the converse. For every platform type 1000 instances were created with the following characteristics: In absolute random platforms, the random values for c_i and w_i vary between 1 and 100, whereas the number of tasks is at least 50. In another test series we make some constraints on the communication and computation powers. In the first one, we decide the communication power to be inferior to the computation power. In this case the values for the communication power vary between 20 and 50 and the computation powers can take values between 50 and 80. In the opposite case, where communication power is supposed to be superior to the computation power, these rates are conversed.

4.2. Distance from the best

We made a series of distance tests to get some information on the mean qualitiy of our algorithms. For this test series we ran all algorithms on 1000 different random platforms of the each type, i.e., homogeneous and heterogeneous, as well as homogeneous communication links with heterogeneous workers and the converse. We normalized the measured schedule makespans over the best result for a given instance. In the following figures we plot the accumulated number of platforms that have a normalized distance less than the indicated distance. This means, we count on how many platforms a certain algorithm achieves results that do not differ more than X% from the best schedule. For exemple in Figure 2(b): The third point of the R-BSA-line significates that about 93% of the schedules of R-BSA differ less than 3% from the best schedule. The figures for platforms with homogeneous communication links and heterogeneous computation powers as well as for platforms with heterogeneous communication links and homogeneous computation powers can be found in [10].

Our results on homogeneous platforms can be seen in Figures 2. As expected from the theoretical results, BBA

and MBBSA achieve the same results and behave equally well on all platforms. R-BSA in contrast shows a sensibility on the platform characteristics. When the communication power is less than the computation power, i.e., the c_i-values are bigger, R-BSA behaves as good as MBBSA and BBA. But in the case of small c_i-values or on homogeneous platforms without constraints on the power rates, R-BSA achieves worse results.

The simulation results on platforms with homogeneous communication links and heterogeneous computation powers consolidate the theoretical predictions: Independently of the platform parameters, MBBSA always obtains optimal results, BBA differs slightly when high precision is demanded. The behavior of R-BSA strongly depends on the platform parameters: when communications are slower than computations, it achieves good results.

On platforms with heterogeneous communication links and homogeneous workers, BBA has by far the poorest results, whereas R-BSA shows a good behavior. In general it outperforms MBBSA, but when the communication links are fast, MBBSA is the best.

The results on heterogeneous platforms are equivalent to these on platforms with heterogeneous communication links and homogeneous workers, as can be seen in Figure 3. R-BSA seems to be a good candidate, whereas BBA is to avoid as the gap is up to more than 40%.

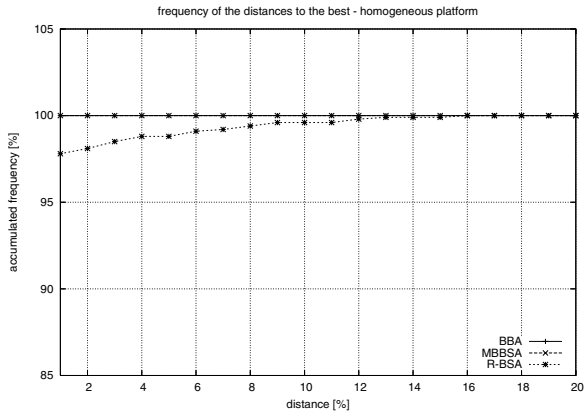
4.3. Mean distance and standard deviation

We also computed for every algorithm the mean distance from the best on each platform type. These calculations are based on the simulation results on the 1000 random platforms of Section 4.2. As you can see in Table 2 in general MBBSA achieves the best results. On homogeneous platforms BBA behaves just as well as MBBSA and on platforms with homogeneous communication links it also performs as well. When communication links are heterogeneous and there is no knowledge about platform parameters, R-BSA outperforms the other algorithms and BBA is by far the worse choice.

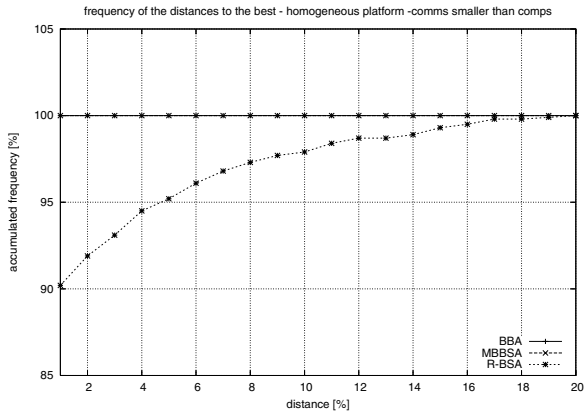
The standard deviations of all algorithms over the 1000 platforms are shown in the right part of Table 2. These values mirror exactly the same conclusions as the listing of the mean distances in the left part, so we do not comment on them particularly. We only want to point out that the standard deviation of MBBSA always keeps small values, whereas in case of heterogeneous communication links BBA-heuristic is not recommendable.

5. Related work

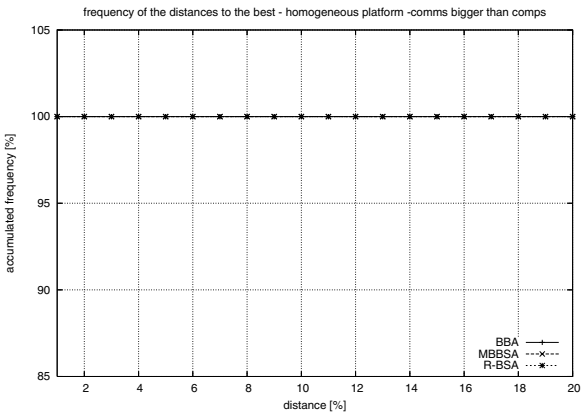
To the best of our knowledge, there are no papers dealing with the same type of data redistribution algorithms which



(a) Homogeneous platform (general case).



(b) Homogeneous platform, faster communicating.



(c) Homogeneous platform, faster computing.

Figure 2. Frequency of the distance to the best on homogeneous platforms.

0-7695-2784-1/07 $20.00

Platform type			Mean distance			Standard deviation		
Comm.	Comp.		BBA	MBBSA	R-BSA	BBA	MBBSA	R-BSA
Hom	Hom		1	1	1.0014	0	0	0.0107
Hom	Hom	$c \leq w$	1	1	1.0061	0	0	0.0234
Hom	Hom	$c \geq w$	1	1	1	0	0	0
Hom	Het		1.0000	1	1.0068	0.0006	0	0.0181
Hom	Het	$c \leq w$	1.0003	1	1.0186	0.0010	0	0.0395
Hom	Het	$c \geq w$	1	1	1.0017	0	0	0.0040
Het	Hom		1.1894	1.0074	1.0058	0.4007	0.0208	0.0173
Het	Hom	$c \leq w$	1.0318	1.0049	1.0145	0.0483	0.0131	0.0369
Het	Hom	$c \geq w$	1.0291	1.0025	1.0024	0.0415	0.0097	0.0095
Het	Het		1.2100	1.0127	1.0099	0.3516	0.0327	0.0284
Het	Het	$c \leq w$	1.0296	1.0055	1.0189	0.0450	0.0127	0.0407
Het	Het	$c \geq w$	1.0261	1.0045	1.0046	0.0384	0.0118	0.0121

Table 2. Mean distance from the best and standard deviation of the different algorithms on each platform type.

can be overlapped by computations (provided that enough data is available locally).

However, REDISTRIBUTION ALGORITHMS have been well studied in the literature. Unfortunately already simple redistribution problems are NP complete [8]. For this reason, optimal algorithms can be designed only for particular cases, as it is done in [13]. In their research, the authors restrict the platform architecture to ring topologies, both uni-directional and bidirectional. In the homogeneous case, they were able to prove optimality, but the heterogenous case is still an open problem. In spite of this, other efficient algorithms have been proposed. For topologies like trees or hypercubes some results are presented in [17].

The LOAD BALANCING PROBLEM is not directly dealt with in this paper. Anyway we want to quote some key references to this subject, as the results of these algorithms are the starting point for the redistribution process. Generally load balancing techniques can be classified into two categories. Dynamic load balancing strategies and static load balancing. Dynamic techniques might use the past for the prediction of the future as it is the case in [4] or they suppose that the load varies permanently [7]. That is why for our problem static algorithms are more interesting: we are only treating star-platforms and as the amount of load to be treated is known a priory we do not need prediction. For homogeneous platforms, the papers in [15] survey existing results. Heterogeneous solutions are presented in [12] or [1]. This last paper is about a dynamic load balancing method for data parallel applications, called the WORKING-MANAGER METHOD: the manager is supposed to use its idle time to process data itself. So the heuristic is simple: when the manager does not perform any control task it has to work, otherwise it schedules.

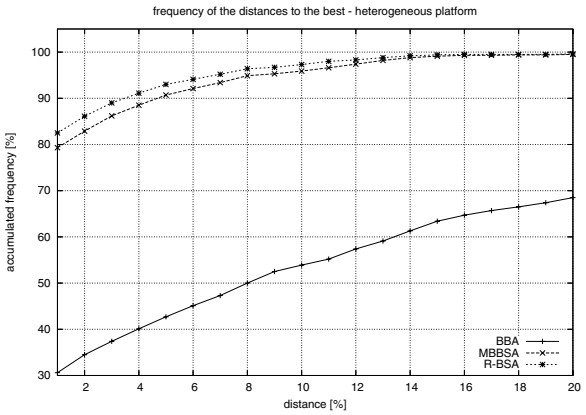
6. Conclusion

We have dealt with the problem of scheduling and redistributing independent and identical tasks on heterogeneous master-slave platforms. Because this problem is NP-complete in the strong sense for completely heterogeneous platforms, we have presented and assessed three heuristics.

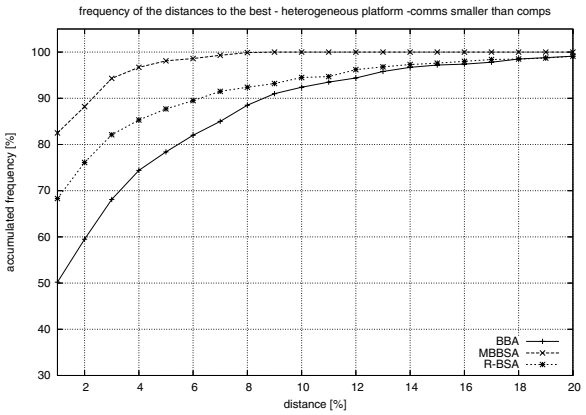
The simulations consolidate the theoretical results of optimality [10]. On homogeneous platforms, BBA is to privilege over MBBSA, as the complexity is remarkably lower. The tests on heterogeneous platforms show that BBA performs rather poorly in comparison to MBBSA and R-BSA. MBBSA in general achieves the best results, it might be outperformed by R-BSA when platform parameters have a certain constellation, i.e., when workers compute faster than they are communicating.

A natural extension of this work would be to derive approximation algorithms, i.e., heuristics whose worst-case is guaranteed within a certain factor to the optimal, for the fully heterogeneous case. However, it is often the case in scheduling problems for heterogeneous platforms that approximation ratios contain the quotient of the largest platform parameter by the smallest one, thereby leading to very pessimistic results in practical situations.

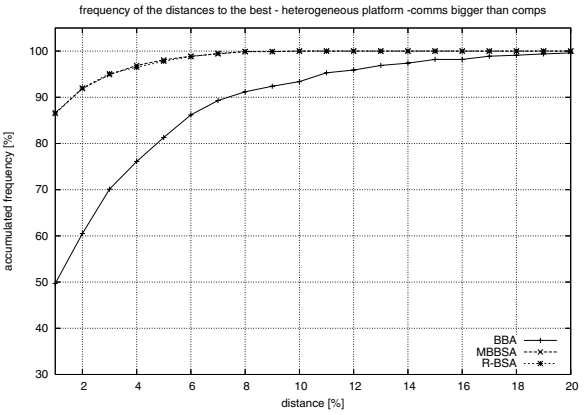
More generally, much work remains to be done along the same lines of load-balancing and redistributing while computation goes on. We can envision dynamic master-slave platforms whose characteristics vary over time, or even where new resources are enrolled temporarily in the execution. We can also deal with more complex interconnection networks, allowing slaves to circumvent the master and exchange data directly.



(a) Heterogeneous platform (general case).



(b) Heterogeneous platform, faster communicating.



(c) Heterogeneous platform, faster computing.

Figure 3. Frequency of the distance to the best on heterogeneous platforms.

References

[1] A. Bevilacqua. A dynamic load balancing method on a heterogeneous cluster of workstations. *Informatica*, 23(1):49–56, 1999.

[2] BOINC: Berkeley Open Infrastructure for Network Computing. `http://boinc.berkeley.edu`.

[3] P. Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2004.

[4] M. Cierniak, M. Zaki, and W. Li. Customized dynamic load balancing for a network of workstations. *Journal of Parallel and Distributed Computing*, 43:156–162, 1997.

[5] Einstein@Home. `http://einstein.phys.usm.edu`.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[7] M. Hamdi and C. Lee. Dynamic load balancing of data parallel applications on a distributed network. In *ICS'95*, pages 170–179. ACM Press, 1995.

[8] U. Kremer. NP-Completeness of dynamic remapping. In *Proceedings of the Fourth Workshop on Compilers for Parallel Computers*, Delft, The Netherlands, 1993.

[9] A. Legrand, L.Marchal, and H. Casanova. Scheduling Distributed Applications: The SIMGRID Simulation Framework. In *CCGrid'03*, pages 138–145, May 2003.

[10] L. Marchal, V. Rehn, Y. Robert, and F. Vivien. Scheduling and data redistribution strategies on star platforms. Research Report 2006-23, LIP, ENS Lyon, France, June 2006.

[11] J. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1), Sept. 1968.

[12] M. Nibhanupudi and B. Szymanski. Bsp-based adaptive parallel processing. In R. Buyya, editor, *High Performance Cluster Computing. Volume* 1*: Architecture and Systems*, pages 702–721. Prentice-Hall, 1999.

[13] H. Renard, Y. Robert, and F. Vivien. Data redistribution algorithms for heterogeneous processor rings. Research Report RR-2004-28, LIP, ENS Lyon, France, May 2004.

[14] SETI. URL: `http://setiathome.ssl.berkeley.edu`.

[15] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and load balancing in parallel and distributed systems*. IEEE Computer Science Press, 1995.

[16] SimGrid. URL: `http://simgrid.gforge.inria.fr`.

[17] M.-Y. Wu. On runtime parallel scheduling for processor load balancing. *IEEE Trans. Parallel and Distributed Systems*, 8(2):173–186, 1997.