Energy-Aware Service Allocation

Damien Borgetto[a], Henri Casanova[b], Georges Da Costa[a], Jean-Marc Pierson[a]

[a]*IRIT, University of Toulouse, Toulouse, France*
[b] *Department of Information and Computer Sciences University of Hawai'i at Mānoa, Honolulu, U.S.A.*

Abstract

In this paper we study the problem of energy-aware resource allocation for hosting long-term services or on-demand compute jobs in clusters, e.g., deployed as part of computing infrastructures. We formalize the problem as three constrained optimization problems: maximize job performance under power consumption constraints, minimize power consumption under job performance constraints, and optimize a linear combination of power consumption and job performance. These problems are NP-hard but, given an instance, a bound on the optimal solution can be computed via a rational linear program. We propose polynomial heuristics for all three problems. Simulation experiments show that in all three cases some heuristics can achieve results close to optimal, i.e., lead to good job performance while conserving energy.

1. Introduction

Commodity clusters have become the most common parallel computing platforms and are mainstream in wide range of settings, including high-performance computing, internet service hosting, and data processing. The scale of deployed clusters ranges from a few hundred nodes to tens of thousands. At large-scale, these clusters represent a large initial investments in terms of hardware. But the largest fraction of their operating cost comes from electrical power consumption, both for powering compute nodes and switches and for providing adequate cooling. Consequently, maintaining near maximum resource utilization is a paramount concern [1]. To avoid resource overprovisioning, the common solution is to consolidate multiple running jobs and/or services on a single physical compute node, possibly powering down unused compute nodes and/or network elements temporarily. A key enabling mechanism for such consolidation is virtual machine technology, used routinely for service hosting and more recently advocated for high-performance computing [2].

While the mechanisms for consolidation are well understood, one challenge is to define metrics that characterize "good" allocations. Another challenge is to design resource allocation algorithms that optimize these metrics. These challenges have been studied in previous work [3, 4, 5, 6, 7, 8]. In particular, Stillwell et al. in [9] have proposed formalizations of the resource allocation and consolidation problem. They define an objective metric called the *yield*, which measures to which extent a job or service receives resource shares that match its effective resource needs. Their work, however, does not consider electrical power consumption. In this paper, using the work by Stillwell et al. as a foundation, we make the following contributions:

1. We formulate the resource allocation problem as a multi-objective optimization problem, with both a yield objective and a power consumption objective;
2. We define three classes of resource allocation problems with three different ways of transforming the multi-objective problem into a single-objective problem;
3. We propose algorithms to solve these problems, some of them coming from the standard literature, some based on the work in [9], and some completely novel;
4. We evaluate our algorithms in simulation and compare them both to each other and to theoretical bounds on optimal solutions.

This paper is organized as follows. Section 2 de-

Email addresses: borgetto@irit.fr (Damien Borgetto), henric@hawaii.edu (Henri Casanova), dacosta@irit.fr (Georges Da Costa), pierson@irit.fr (Jean-Marc Pierson)

scribes the resource allocation problem, notations, and objectives, as well as the three studied subproblems. Section 3 proposes resource allocation algorithms. Simulation results are presented in Section 4 and discussed in Section 5. Section 6 reviews related work and Section 7 concludes the paper with a summary of results and future work directions.

2. The Resource Allocation Problem

2.1. Model and Notations

We consider a cluster that consists of H compute nodes, or hosts. We abstract each host as a provider of two resources: CPU cycles per time unit and RAM space. Using virtualization, precise and arbitrary shares of these resources can be allocated to virtual machine (VM) instances. We assume that all hosts provide the same amount of CPU and RAM resources, and thus we only talk of resource shares as fractions of total host resources, i.e., numbers between 0 and 1. Host h, $h = 1, \dots, N$, consumes C_h^{max} Watts when its CPU is fully utilized. If a fraction f of its CPU resources is utilized, then the power consumption is $C_h^{min} + f \times (C_h^{max} - C_h^{min})$, where C_h^{min} corresponds to the power used when the host is idle ($C_h^{min} < C_h^{max}$). While not fully realistic, this simple model is corroborated by the experiments reported in [10]. If a host is unused, then we assume that it is simply powered off and has a zero power consumption.

We consider N services, or jobs, that run on the cluster. As in [9], we focus on an off-line resource allocation problem with a fixed set of continuously executing jobs that have constant resource needs. In practice, jobs could arrive in or depart from the system, and could have resource needs that evolve throughout time. We leave such an on-line scenario for future work. But we note that if good and fast algorithms are identified for the off-line problem, these algorithms form a sound basis for developing on-line resource allocation strategies (as seen, for example, in [11]). We consider sequential jobs, meaning that each job can be encapsulated inside a single VM instance. Extension to multi-instance jobs is immediate [9].

Job i is defined by a *CPU need*, α_i, and a *RAM requirement*, m_i. The CPU need of a job is the CPU share it would use on a host that is dedicated to that job. The job cannot make use of a larger share of the CPU (e.g., due to time spent idle, spent doing I/O or network operations). If a job is allocated a CPU share that is lower than its CPU need then it operates at reduced speed. For instance, if a job has a CPU need of 50% but is running within a VM instance that is constrained to using at most 10% of the CPU resources of a host, then the job operates at one fifth of its maximum speed. Using the terminology in [9], we call the ratio of the job's CPU need to the job's allocated CPU share the job's *yield*. In the above example, the job has a yield of 0.2. A yield of 1.0 means that the job receives all the CPU resources it can use, while a yield of 0.0 means that the job cannot run (denoting a failure of the resource allocation process). The RAM requirement corresponds to share of RAM space necessary for the job to run. The job cannot make use of a larger share, but cannot run with a smaller share.

The resource allocation problem is trying to allocate each jobs to a host and giving it a fraction of the host's resource. The resolution of this problem is a success when every jobs has been mapped to a host within the capability of the system. On the contrary, failure of the allocation is when at least one job could not be placed. We say that a resource allocation algorithm fails when the algorithm can't provide a successful allocation.

2.2. Constraints

In this section we derive constraints that form the basis for a linear program formulation of the resource allocation problem. This program extends that in [9] to account for power consumption. We first define the following variables. e_{ih} is a binary variable that takes value 1 if job i is allocated to host h, and takes value 0 otherwise. α_{ih} is a rational variables that denotes the CPU fraction allocated to job i on host h. Finally, p_h is a binary variable that is set to 1 if host h is powered on. Our constraints for these variables are as follows:

$$\begin{array}{rcr}
\forall i, h & e_{ih} \in \{0,1\} & (1) \\
\forall i, h & \alpha_{ih} \in \mathbb{Q} & (2) \\
\forall h & p_h \in \{0,1\} & (3) \\
\forall i & \sum_h e_{ih} = 1 & (4) \\
\forall i, h & 0 \leq \alpha_{ih} \leq e_{ih} & (5) \\
\forall i, h & p_h \geq \alpha_{ih} & (6) \\
\forall h & p_h \leq \sum_i e_{ih} & (7) \\
\forall h & \sum_i \alpha_{ih} \leq p_h & (8) \\
\forall h & \sum_i e_{ih} m_i \leq p_h & (9) \\
\forall i & \sum_h \alpha_{ih} \leq \alpha_i & (10) \\
 & & (11)
\end{array}$$

Constraints 1-3 define the range of the variables. Constraint 4 states that job is allocated to a single host, meaning that for a given i only one e_{ih} value is non-zero. Constraint 5 states that a job can consume CPU resources only on the host to which it is allocated, meaning that for a given i only one α_{ih} value is non-zero. Constraint 6 states that if a job consumes CPU resources on a host then that host must be powered on. Constraint 7 states that if no job is allocated to a host then that host is powered off. Constraint 8 states that the CPU resources of a host that is powered on are not to be exceeded, and that no CPU resources are consumed on a host that is powered off. Constraint 9 states that the RAM resources of a host that is powered on are not to be exceeded, and that no RAM resources are consumed on a host that is powered off. Finally, Constraint 10 states that a job never receives a CPU share that is larger than its CPU need.

2.3. Objectives

We can define multiple objective functions that define optimization problems subject to the constraints defined in the previous section. In [9], the authors use the *minimum yield* over all jobs, Y, which is defined by the following additional constraint:

$$\forall i, \quad \sum_h \frac{\alpha_{ih}}{\alpha_i} \geq Y \ . \tag{12}$$

The objective is to maximize Y, which is a good way to capture notions of both fairness and performance.

While the yield relates to the levels of performance experience by the jobs, an important aspect of a resource allocation is its power consumption. We define this power consumption E, expressed in Watts, as:

$$E = \quad \sum_h C_h^{min} p_h + \sum_h (C_h^{max} - C_h^{min}) \sum_i \alpha_{ih} \ . \tag{13}$$

In other words, the power consumption is equal to the idle power consumption of all hosts that are powered on plus the additional power consumption of these hosts due to their CPU load. An objective is then to minimize E.

We have thus defined a performance-related objective (maximize Y) and a power-related objective (minimize E). Multi-objective optimization problems are notorious for being difficult. Therefore, in the following three sections, we define single-objective versions of the problem.

2.4. The BoundedPower *problem*

We define BoundedPower by adding the constraints that E is bounded above by a user-provided constant E^{bound}. This constant represents the maximum affordable power budget for the operation of the cluster. The optimization problem then becomes a Mixed Integer Linear Program (MILP), with the objective of maximizing Y. Relaxing all integer variables in the MILP to be rational, we obtain a relaxed rational linear program that (i) can be solved in polynomial time; and (ii) provides a theoretical upper bound on the optimal Y value. However, the relaxation of all the integer variables comes with a cost, that is the resulting allocation not being realistic. For instance, one job can be spread over several hosts, and a host can be half switched-on.

2.5. The BoundedYield *problem*

We define BoundedYield by adding the constraint that Y is bounded below by a user-provided constant Y^{bound}. This constant represents a guaranteed minimum level of job "satisfaction," where satisfaction is defined as the achieved fraction of the job's CPU need. As above the optimization problem is a MILP with the objective of minimizing E, and a theoretical lower bound on the optimal E value can be computed in polynomial time.

2.6. The MixedObjective *problem*

An alternative to bounding one objective and optimizing the other is optimizing a linear combination of the two, e.g., optimizing $Z = \lambda Y + (1 -$

$\lambda)(1/E)$, with λ is between 0 and 1. We call this problem MixedObjective. This problem definition has two drawbacks. The main drawback is that the value of λ must be chosen by the user. While a higher value indicates a larger weight for performance than for power consumption, the tradeoff is not precisely defined. Also, there is no guarantee that the obtained solution, defined by achieved minimum yield and power consumption values $\hat{Y}$ and $\hat{E}$, is Pareto optimal. However, this issue can be addressed by solving BoundedPower (resp. BoundedYield) using $\hat{E}$ (resp. $\hat{Y}$) as the bound on E (resp. Y). Another problem is that the units and magnitudes of Y and E are not commensurate. Since Y takes values between 0 and 1, it is necessary to normalize E so that it too takes values between 0 and 1. This can be done by defining the objective as maximizing $Z = \lambda Y + (1-\lambda)(1 - E/E^{max})$, where E^{max} represents the maximum power consumption of the platform.

A naïve definition of E^{max} would be $\sum_h C_h^{max}$. While this is the maximum possible power consumption of the platform, it may not be a good choice for normalizing the E value. Considering a trivial allocation problem for a single job, and assuming for the sake of this discussion that all hosts have the same C^{max} value, E/E^{max} would then be smaller than $1/H$. In general, with this definition of E^{max}, the range of values for E/E^{max} could be orders of magnitude smaller than that for Y, making the linear combination of the two inadequate. E^{max} should instead be a reasonable upper bound on the power consumption of a reasonable resource allocation for the workload at hand.

To compute E^{max}, we first solve the relaxed linear program in Section 2.4 setting $E = \sum_h C_h^{max}$. This linear program produces, in polynomial time, a bound on the optimal value for Y. Let us use Y^{bound} to denote this value. We then perform a binary search for E^{max} over the range between 0.0 and $\sum_h C_h^{max}$. For each candidate E^{max} value in this binary search, we solve the relaxed rational linear program mentioned in Section 2.4, fixing $E = E^{max}$. If this program produces a Y value that is no lower than Y^{bound}, then we attempt a lower E^{max} value. Otherwise, we attempt a higher value. We use a 0.01 threshold to define the precision of this binary search. Our rationale is that E^{max} is the smallest upper bound on E for which the theoretical bound on the optimal yield is no lower than that obtained assuming the highest possible power consumption.

Note that feasible allocations can have power consumptions higher than E^{max}. For instance, consideri 2 completely identical hosts (with identical and strictly positive C^{min} and C^{max} values), and 2 jobs each with a CPU need of 50% and a memory requirement of 50%. In this case, $E^{max} = C^{max}$. However, a valid allocation is one in which each job is on a different host, leading to a power consumption of $2(C^{min} + .5 \times C^{max}) > E^{max}$.

Consequently, to ensure that both terms in our linear combination of the objectives are always between 0 and 1, we redefine our maximization objective as:

$$Z = \lambda Y + (1-\lambda)(1 - X) ,$$

where $X = \min(1, E/E^{max})$. In this way, we have obtained a MILP formulation for MixedObjective. Like for the two preceding problems, a theoretical bound on the optimal Z value can be obtained by relaxing integer variables to be rational.

3. Algorithms

3.1. Greedy

We first propose algorithms that build on well-known greedy algorithms for vector packing, namely First Fit and Best Fit.

To solve the BoundedYield problems, we propose two First Fit algorithms. The Greedy_BoundedYield_1 algorithm sorts the jobs by decreasing m_i, while the Greedy_BoundedYield_2 algorithm sorts the jobs by decreasing α_i. Assuming that all jobs have a yield equal to Y^{bound}, both algorithms place each job on the host with the lowest C^{max} that can accommodate the job's resource needs. The Greedy_BoundedYield_3 and Greedy_BoundedYield_4 are the Best Fit versions of Greedy_BoundedYield_1 and Greedy_BoundedYield_2, respectively.

To solve BoundedPower, we also propose two First Fit algorithms that aim at favoring hosts with the lowest power consumptions. The Greedy_BoundedPower_1 algorithm sorts the hosts by decreasing m_i, and the Greedy_BoundedPower_2 sorts the jobs by decreasing α_i. Initially, both algorithms assume that all jobs have a yield $Y = 1$, and attempts to place all jobs if possible. The algorithms then iteratively decrease non-zero α_{ij} values by 0.01 in a round robin fashion over all jobs. This is done until all jobs can be placed and the allocation's power consumption reaches the specified energy bound. If a job is given a yield of 0 before these conditions are reached then resource allocation fails. GREEDY_BOUNDEDPOWER_3 and GREEDY_BOUNDEDPOWER_4 are the Best Fit versions of GREEDY_BOUNDEDPOWER_3 and GREEDY_BOUNDEDPOWER_4, respectively.

The GREEDY_MIXEDOBJECTIVE_1 and GREEDY_MIXEDOBJECTIVE_2 algorithms solve the MIXEDOBJECTIVE problems by building on the above two algorithms for solving BOUNDEDPOWER. They first calculate $H_Z = max(1, \lceil(1 - \lambda) \times (\sum_i \alpha_i)\rceil)$, which gives the theoretical number of hosts that is needed to achieve a resource allocation with a given λ value. The energy bound is then calculated using the H_Z first hosts in the host list (the hosts are sorted by increasing C^{max}). This energy bound is used to solve BOUNDEDPOWER using either GREEDY_BOUNDEDPOWER_1 or GREEDY_BOUNDEDPOWER_2.

3.2. EA-ResAlloc Algorithms

In [12] we proposed a resource allocation heuristic based on the work in [9] that aims at addressing two antagonist objectives: maximizing the job yield and minimizing power consumption. This heuristic relies on the *energy-aware yield* of a job $i \in [1..J]$ allocated to host $h \in [1..H]$, defined as:

$$YE_{ih} = \frac{\left[\sum_{h=1}^{H}(\frac{\alpha_{ih}}{\alpha_i})\right]^{1-k}}{\left[\gamma\delta C_{ih}+(1-\gamma)(A_h(1-\sum_{i'=1,i'\neq i}^{J}(\alpha_{i'h})))\right]^{k}}$$

$$= \frac{(Y_{ih})^{1-k}}{(E_{ih})^{k}}$$

The Y_{ih} part is simply the yield of job i. The E_{ih} part is correlated to the allocation's power consumption. The k parameter thus allows a tradeoff between yield and power consumption.

δC_{ih} represents the contribution of job i to the power consumption of host h without taking into account the consumption of host h while idle. This term favors the placement of jobs on energy efficient hosts.

A_h is the "attractiveness" of host h, a smaller value denoting a more attractive host. A host is more attractive than another if, without taking into account any other jobs, adding a job requires less energy on it. The factor $A_h(1 - \sum_{i'=1,i'\neq i}^{J}(\alpha_{i'h}))$ is used to control the grouping of jobs on attractive hosts. It helps choosing the most attractive host when no host is already loaded, and choosing an already loaded host when there is one. In other words, this term is used to favor workload consolidation.

The γ parameter thus allows a tradeoff between placement (choosing the most energy-efficient hosts) and consolidation (aggregating jobs on a small amount of hosts). Note that both tend to decrease energy consumption.

The energy-aware yield YE_{ih} is used by the heuristic to conciliate 3 different goals:

- maximizing job yield;
- placing jobs on energy efficient hosts; and
- aggregating jobs on a reduced number of hosts in order to power down as many hosts as possible.

This metric is used in a task allocation heuristic using bin-packing, taking into consideration also the memory required by jobs and available on the hosts. Simulation results in [12] have shown that using the energy-aware yield leads to good results, namely reducing the energy consumption while achieving good performances (in terms of average job yields). In this work, we build on this heuristic so that it can be applied to solve the BOUNDEDYIELD, BOUNDEDPOWER, and MIXEDOBJECTIVE problems.

For the BOUNDEDPOWER problem, we simply do a binary search on the parameter k in the energy-aware yield definition, with the constraint that the power consumption of the resulting resource allocation is under E^{bound}. This amounts to finding the lowest feasible value for k.

To solve the BOUNDEDYIELD problem, we allocate resources with an objective yield of Y^{bound} and with $k = 0$.

The MIXEDOBJECTIVE problem is solved with the energy-aware yield metric as it is, as it was designed specifically for the bi-criteria approach of the problem. As k is designed to make a tradeoff between energy consumption and job performance, we use $k = \lambda$ (k being the energy/performance tradeoff parameter of the *energy-aware yield* metric, λ being the tradeoff parameter of the MIXEDOBJECTIVE problem).

For each problems, we use $\gamma = \min(\frac{H}{J}, 1)$. For γ values close to 1, i.e., when the system is not overloaded, the heuristic will mostly attempt to consolidate the workload. For γ values close to 0, the heuristic will mostly attempt to perform good job placement.

3.3. Bounds on optimal

Like the resource allocation problem in [9], our three problems are NP-hard via trivial reduction to the bin packing problem. Their MILP formulations cannot be solved in polynomial time due to the presence of integer variables. We can therefore compute optimal solutions only for small problem instances. In our results we compute optimal solutions only for instances with 4 hosts and at most 12 jobs. We term this solution MILP.

In a view to comparing our algorithms to the optimal for larger instances we consider relaxed versions of our problems in which the integer variables of the MILP formulation are made rational. The solution can then be computed in polynomial time in practice. In general, this solution is not a valid solutin of the original MILP because it may spread a job across multiple hosts. Nevertheless, it is useful because it provides a bound on the optimal solution (either in terms of yield, or power consumption, or of a combined objective). We term this solution LPBOUND. If our algorithms achieve solutions close to this bound, then their solutions are even closer to the optimal.

We use the free and open source GLPK [13] solver to compute MILP and LPBOUND. Commercial solvers, such as ILOG CPLEX[14], would allow us to compute MILP for larger instances, but LPBOUND is sufficient to evaluate the efficacy of our algorithms in an absolute sense.

4. Experimental Results

4.1. Methodology

In this section we detail our experimental methodology. To pick representative values for C^{min} and C^{max} we monitored power consumption on the Grid5000 [15] clusters in Lyon, Toulouse and Grenoble. We found a mean idle consumption of around $120W$ and full load consumption of $190W$, both with a standard deviation of respectively 20 and 30. Based on our measurements, we construct problem instances with hosts whose C^{min} values fall between 100 and 140, and whose C^{max} values fall between 160 and 220. Actual values are sampled uniformly from these intervals, as we didn't take interest in the effective distribution of power amongst the hosts of the cluster. In our instances, job CPU needs are sampled from a normal distribution with a mean of 0.33 and a standard deviation of 0.5 which we truncate to rule out negatives values and scale to match the mean. Given the lack of an accept model for CPU needs of real workloads, we have opted for this simple model that leads to a diverse job mix. We generated memory needs for the jobs between 2% and 95% following a Pareto distribution of order 6, since typically only a few jobs consume a lot of memory. Note that, in our model, CPU needs and memory needs are independent. This means that a job that has a high CPU need does not necessarily have a high memory need. Using the above distributions we generate 100 random instances for each H hosts ($H = \{4, 6, 8, 16, 32, 64\}$) and J jobs ($J = \{H, 2H, 3H\}$) combination.

For each instance we solve the BoundedPower, BoundedYield and MixedObjective problems with the algorithms in Section 3, computing the MILP solution only for small instances as explained in Section 3.3. For BoundedYield (resp. BoundedPower) we randomly generate the bounds between 50% and 100% of $max(1, \frac{\sum_i \alpha_i}{H})$ (resp. $\sum_h C^{max}$). For the MixedObjective problem, we chose λ valus of 0.2 (foxus on energy), 0.5 (balanced focus) and 0.8 (focus on performance).

For each of the produced solutions we compute Y (the minimum yield), E (the power consumption), and the number of hosts that are powered up. We give mean and standard deviations of these values computed over the 100 samples for selected H and J combinations. Note that while most of our random instances have solutions, we do not guarantee their feasibility. Algorithm failure rates reported in what follows include failures for (a few) instances that are not solvable.

4.2. The BoundedYield *problem*

In this section we evaluate our algorithms when used to solve BoundedYield. For this problem, if the bound on the minimum yield cannot be achieved, then the algorithm fails. We only present results for the energy, E, since all algorithms achieve the same yield when they succeed. We discuss the failure rates of each algorithm as well as the number of hosts they use.

4.2.1. Results for small instances

We first discuss results for instances with $H = 4$ hosts, for which the optimal solution, MILP, can be computed. Figure 1 plots the mean energy (and

Table 1: Variables summary

Variable	**Name**	**Range**
α_{jh}	CPU allocated to job j on host h	[0,1]
α_j	CPU required by job j	[0,1]
m_j	Memory required by job j	[0,1]
e_{jh}	Presence of job j on host h	$\{0 \mid 1\}$
p_h	host h is powered on	$\{0 \mid 1\}$
C^{min}	Min host consumption	$[0,\infty]$
C^{max}	Max host consumption	$[0,\infty]$
Y	Minimum yield	[0,1]
E	Energy Consumption	$[0,\infty]$
λ	Tradeoff parameter	[0,1]

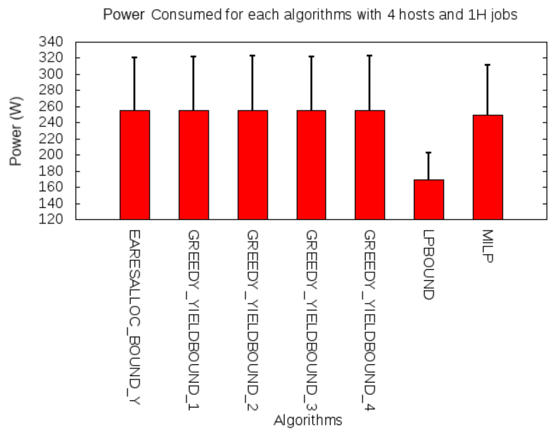

Figure 1: Mean energy consumed for each algorithms with 4 hosts and 4 jobs

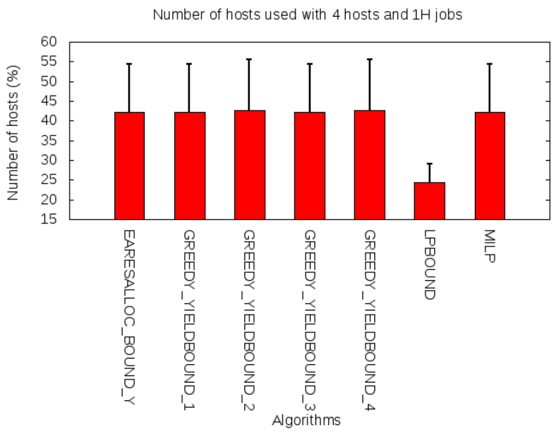

Figure 2: Mean number of hosts used for each algorithms with 4 hosts and 4 jobs

standard deviation) of the resource allocation for the algorithms. We see that the EA-ResAlloc algorithm and the greedy algorithms perform similarly such a small number of hosts, and close to MILP. LPBOUND, because it is a rational (unattainable) bound on the optimal, always outperforms all other solutions since it violates integer constraints.

Figure 2 shows the average and standard deviation of the number of hosts used in the allocation produced by our algorithms when $H = 4$. On average only around 40% of the hosts (i.e., 1.6 host) are used, for more than 50% energy reduction compared to no workload consolidation. In this scenario, the hosts are far from overloaded. Finding a low-energy configuration by powering hosts down is relatively straightforward, as seen in our results. Nevertheless, this experiment shows the importance of having power management algorithms, and that even simple algorithms can be effective. Finally, note that LPBOUND uses on average less than 25% of the hosts, meaning that its solutions amount to powering on only a fraction of a host, which is not possible in practice.

4.2.2. Results for all instances

Table 2 shows the mean energy consumed by all algorithms, computed over all instances, along with the relative percent difference with LPBOUND. We can see that EAResAlloc_Bound_Y performs the best on average, but its competitors are still close to the bound, and thus close to the optimal.

To gain more insight into the results, Figure 3 plots the average energy consumed by each algorithms, grouped by number of hosts in the instance. We only have MILP results for the instances with 4 hosts, since computing it for instances with 6

Table 2: Energy consumed for each algorithms

Algorithm	**Energy**	**stddev**	**% bound**
LPBOUND	1908	2221	0%
EARESALLOC_BOUND_Y	2090	2329	+9.5%
GREEDY_YIELDBOUND_1	2199	2485	+15.2%
GREEDY_YIELDBOUND_2	2145	2396	+12.4%
GREEDY_YIELDBOUND_3	2203	2502	+15.4%
GREEDY_YIELDBOUND_4	2143	2388	+12.3%

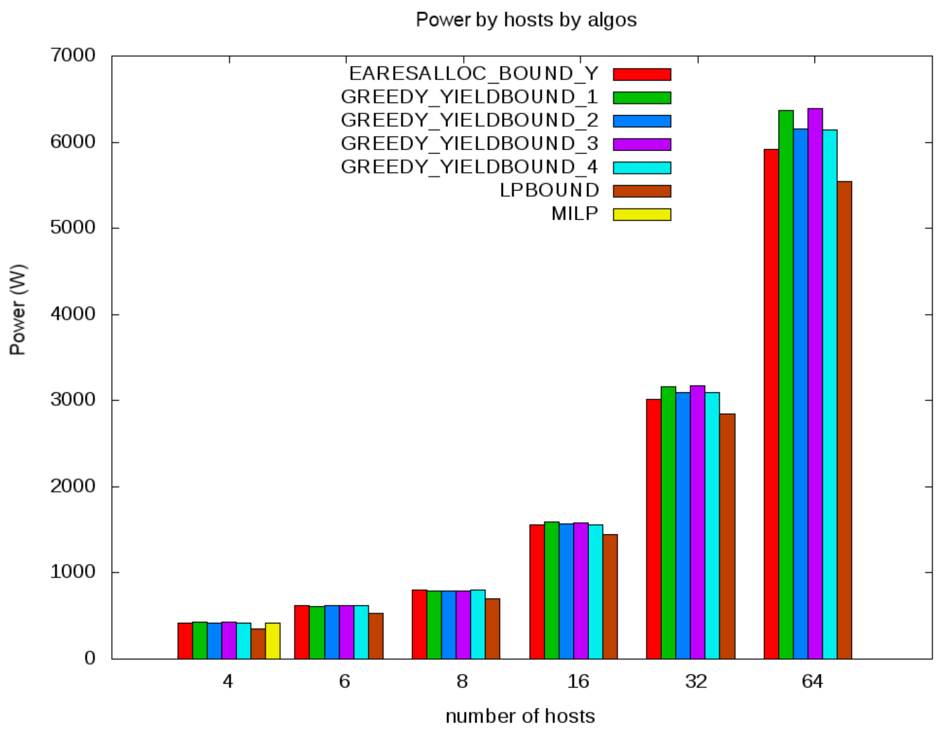

Figure 3: Energy consumed by each algorithm for each number of hosts

hosts and above takes prohibitive amounts of time. As expected, LPBOUND and MILP achieve the smallest average energy consumption. For small, number of hosts (4, 6, and 8), our heuristics all lead to comparable results, as seen earlier in the case of 4 hosts. The EA-ResAlloc_Bound_Y algorithm begins to behave differently from the greedy algorithms for instances with over 16 hosts. For the largest instances it achieves the best results, giving allocation of an average of 5,900 Watts. For these instances, EA-ResAlloc_Bound_Y is approximately 6% away from LPBOUND, while the Greedy_BoundedYield_4, which is the best of the greedy algorithms for these instances, is 10% away from the bound.

Figure 3 aggregates results for multiple numbers of jobs, but the results could be different for $J = 1H$, $J = 2H$ or $J = 3H$ jobs. As expected, all algorithms perform better in the $J = 1H$ case. The $J = 2H$ (resp. $J = 3H$) case is more CPU-intensive, since the average sum of the CPU needs is equal to 66% (resp. 100%) of the computational capability of the hosts, without taking into account the bound on the minimum yield. As the ratio between the number of jobs and the number of hosts grows, the average energy consumed is closer to the rational bound. This is because there is less flexibility when allocating resources, and thus less opportunity for energy reduction. Nevertheless, the ranking of the algorithms seen in Figure 3 is unchanged

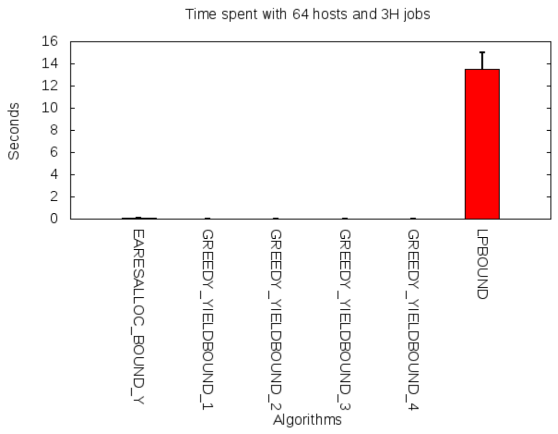

Figure 4: Mean execution time of the all algorithms for 64 hosts and 192 jobs

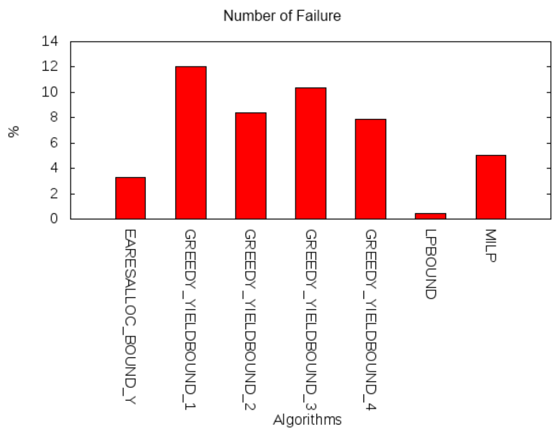

Figure 5: Percent of failure of the algorithms

when examining results for particular numbers of jobs.

4.2.3. Execution time and failure rate

Figure 4 plots the mean execution time of all algorithms. We can see that apart from the LP-BOUND algorithm, which takes more than 12 seconds on average, all others algorithms are fast. Their average execution times go from 67 milliseconds for EAResAlloc_Bound_Y to 1 millisecond for the greedy algorithms. Furthermore, we created an instance with 500 hosts and 1500 jobs and found that EAResAlloc_Bound_Y produces a solution in 1 second. We conclude that our algorithms are sufficiently fast and scalable to be employed in practice. For comparison, the MILP solutions for 4 hosts and 12 jobs is computed in 1.5 seconds on average.

The mean failure rates of the algorithms are shown in Figure 5. The EAResAlloc_Bound_Y algorithm fails for about 3% of the instances. The Greedy_BoundedYield_1, which is the algorithm that fails the most, has a failure rate of 12%. The Greedy_BoundedYield_4 algorithm, which is the best of the greedy algorithms, is also the greedy algorithm that fails the least with a 8% failure rate. These failures occur mostly for instances with $J = 3H$ jobs, as these instances are the hardest to solve (and some of them may not be feasible). Also, expectedly, the rational bound LPBOUND almost never fails. The only failure is when, for example, the sum of the memory needs exceed the hosts capacity. Again, this is because we do not ensure that all our instances are feasible.

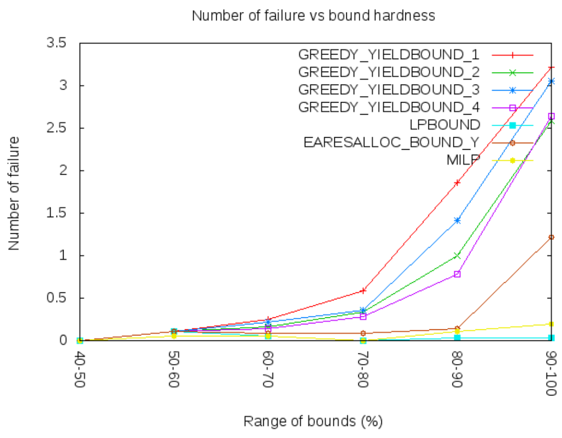

Figure 6: Percent fail per bound hardness

Figure 6 plots failure rate vs. "bound hardness," which we define as the relative percentage difference between the specified yield bound and the bound on the maximum yield that can be achieved, in this case 1.0. As stated in Section 4.1, we only generated bound values between 50% and 100%, hence the plotted range in the figure. Each point represents the failure rate for the corresponding range. The sum of all the points of this graph corresponds to the results shown in Figure 5. The clear, and expected, trend is that the harder the bound, the higher the failure rates. For example, for a bound between 90% and 100% of the maximum, the greedy algorithms fail for about 3% of the instances. The main observation here is that the EAResAlloc_Bound_Y algorithm is significantly more robust than the greedy algorithms across the board.

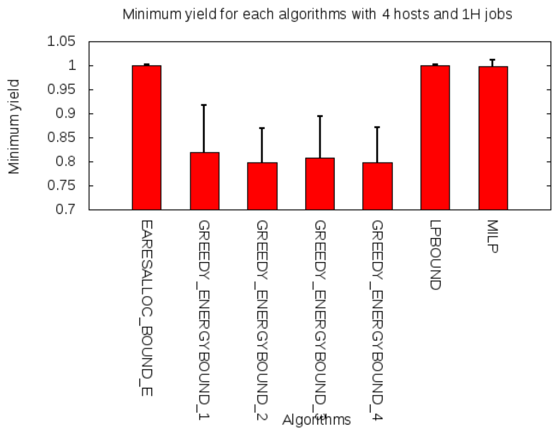

Figure 7: Mean minimum yield for 4 hosts and 4 jobs

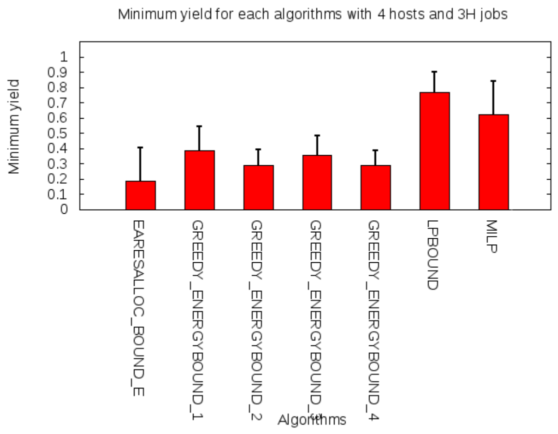

Figure 8: Mean minimum yield for 4 hosts and 12 jobs

4.3. *The* BoundedPower *problem*

4.3.1. *Results for small instances*

A fundamental difference between this problem and BoundedYield is that the bound on the energy consumption caps the number of hosts that can be used in the allocation. Figure 7 shows the performance of the algorithms for $H = 4$ hosts and $J = 4$ jobs, including the MILP solution. Unlike for BoundedYielf, there are striking differences between the algorithms. Intuitively, the algorithms here have to make smarter decisions than just selecting hosts greedily. Since the greedy algorithms first compute an allocation and then try to meet the energy bound using this allocation as a starting point, the minimum yield their produce ends up being reduced drastically. This explains why the performance of the greedy algorithms is as far as 20% lower than the optimal solution. By contrast, the EA-ResAlloc_Bound_E algorithm performs reasonably well even though it does not achieve the optimal solution on all instances. The reason is that our metric is centered on both placement effects (placing the jobs where they will consume the least amount of energy) and consolidation effects (reducing the number of hosts that are powered on). For such small instances, our heuristics turns out to achieve more consolidation than placement, at times leading to sub-optimal allocations.

The poor performance of the greedy algorithms worsens as the the ratio between the number of jobs and the number of hosts increases. This is seen in Figure 8, which is for $J = 12$ jobs, in which case the average minimum yield drops up to to 40% of the optimal solution. The EA-ResAlloc_Bound_E

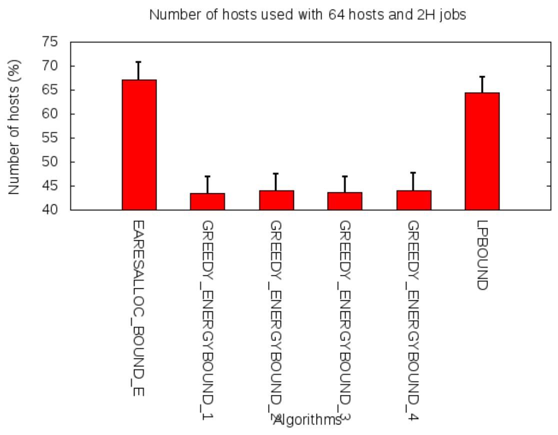

Figure 9: Number of hosts used for 64 hosts and 128 jobs

does not perform well in such cases either. Esentially, there is rarely room for consolidation and, when there is, the minimum yield drops significantly. We can also see that this algorithm has the largest standard deviation, which is due to a higher variability due to a larger number of allocation failures.

4.3.2. *Results for all instances*

Figure 10 plots the average minimum yield for each algorithm, grouped by number of hosts. As expected, the best results are for MILP and LPBOUND. Among the other algorithms, we can see that EA-ResAlloc_Bound_E leads to significantly higher performance on average. Although this algorithm displays poor performance in scenarios with few hosts and a relatively large number of hosts (see Figure 8 for $H = 4$ and $J = 12$), in

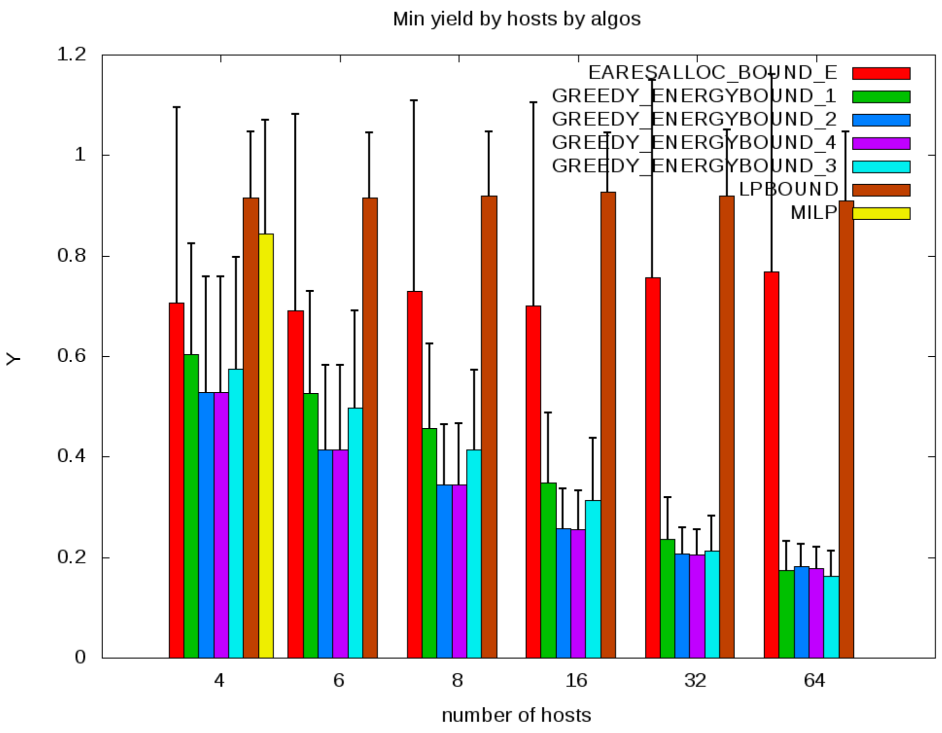

Figure 10: Minimum yield by algorithms by number of hosts

less oversubscribed instances it leads to good performance.

To gain more insight into the difference between EA-ResAlloc_Bound_E and the greedy algorithms, Figure 9 plots the number of hosts used by the algorithms for instances with H = 64 hosts and J = 128 jobs. We see that EA-ResAlloc_Bound_E algorithm uses more hosts on averages than the greedy algorithms. This is explained as follows. On the one hand, EA-ResAlloc_Bound_E tries to achieve good consolidation but also high minimum yield, thus tending to have a high number of hosts. On the other hand, the greedy algorithms tend to favor consolidation, which is detrimental to job yield, but that results in a low number of hosts powered on.

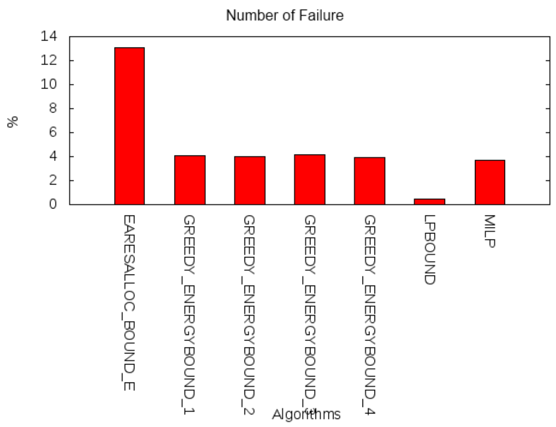

Figure 11: Failure rate of the algorithms

4.3.3. Execution time and failure rate

Figure 13 plots the execution times of the algorithms for 64 hosts and 192 jobs. As we can see, the EA-ResAlloc_Bound_E algorithm takes much longer to compute an allocation than the greedy algorithms. This is because multiple allocations are produced in an attempt to achieve the best energy-aware resource allocation (as part of two binary searches). Consequently, computing on instances of 500 hosts and 1500 jobs would require more than one minute. In such situations, one must then revert to using one of the greedy algorithms.

Figure 11 plots the failure rate of all the algorithms. The main observation is that the EA-ResAlloc_Bound_E algorithm has the highest

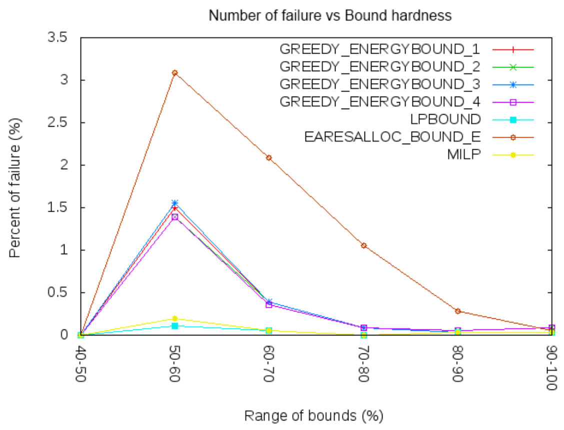

Figure 12: Failure rate vs. bound hardness

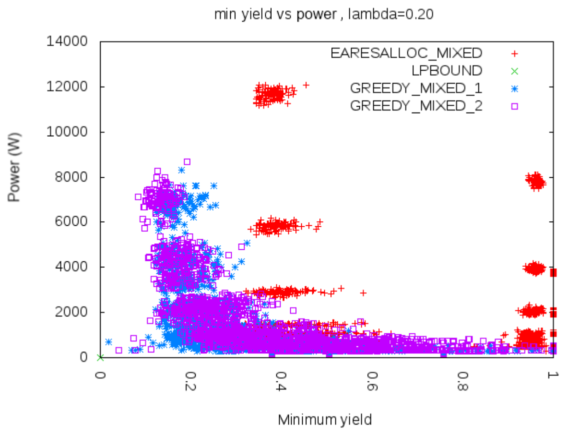

Figure 14: Minimum yield vs. power consumed for $\lambda = 0.2$

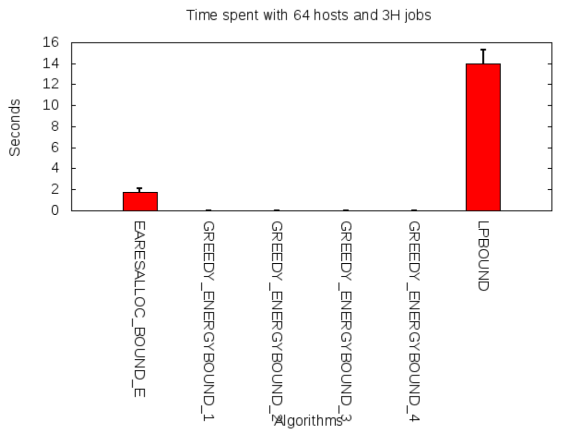

Figure 13: Execution time of the algorithms for 64 hosts and 192 jobs

failure rate, at about 13%. This is because this algorithm does not favor consolidation, leading to resource allocation failures in oversubscribed scenarios. Figure 12 plots the failure rate versus the bound harness (as in Figure 6). This figure shows that the hardness of the bound on the power consumption has an effect on the greedy algorithms as well as on the EA-ResAlloc_Bound_E algorithm. As opposed to Figure 6, a smaller bound percentage for the energy consumption is harder. For example, a bound percentage of 50% on the energy consumption means that the system should not consume more than $0.50 \times \sum_h (C_h^{max})$, which is hard to achieve. We can see here that a hard bound between 50% and 70% gives almost the same failure rate for the greedy algorithms and double for EA-ResAlloc_Bound_E. However, if the failure rate drops for the greedy for easier bounds, the EA-ResAlloc_Bound_E algorithm still fails on a significant number of instances. Most failures come from instances with $J = 3H$, which, as discussed in Section 5, is due to the imbalanced workload consolidation achieved by this algorithm.

4.4. The MixedObjective *problem*

For this problem, it is difficult to compare the algorithms because they do not optimize the same metric. For instance, the EA-ResAlloc_Mixed algorithm optimizes the metric described in Section 3.2, while LPBOUND and MILP optimize the metric described in Section 2.6. As the metrics are not comparable and the problem itself is multi-criterion, we cannot compare the results in terms of performances, as one algorithm could favor the power consumption over the yield and another could do the opposite. Consequently, we only compare what is comparable, i.e., the behavior of the algorithms with different λ values.

Figure 14 plots the results of the algorithms for all the instances, with $\lambda = 0.2$. Such a value for λ favors the increase of the minimum yield over the reduction of the power consumption. We can distinguish the clusters induced by the variation of the number of hosts among our instances, which increases the energy consumption. We can note here that increasing the number of hosts by a factor two does not necessarily increase the power consumption of the resources allocation twofold, for instance for the Greedy_Mixed_1 algorithm. On the one hand we can see that the EA-ResAlloc_Mixed algorithm leads overall to a higher power consump-

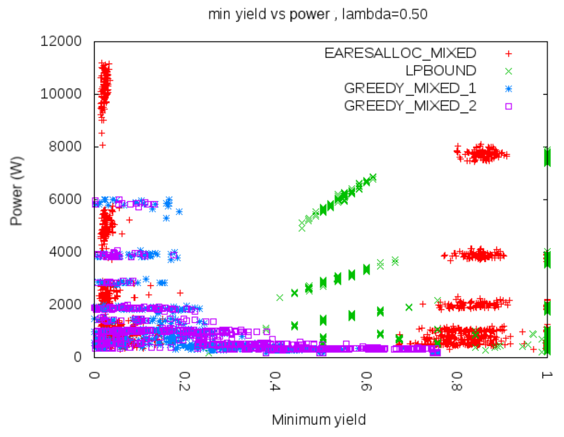

Figure 15: Minimum yield vs. power consumed for $\lambda = 0.5$

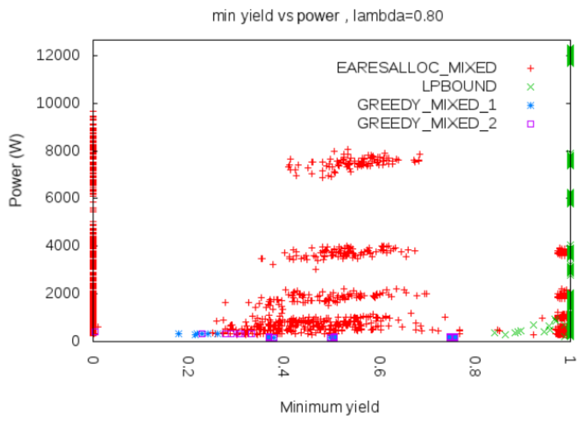

Figure 16: Minimum yield vs. power consumed for $\lambda = 0.8$

tion than the GREEDY_MIXED_1 algorithm. On the other hand, the minimum yield achieved is greater with EA-RESALLOC_MIXED. When $\lambda = 0.2$, we want to favor the minimum yield over the energy consumption, which is closer to the use case when the minimum yield is the solve objective.

Figure 15 plots the values for $\lambda = 0.5$, which means that we want to optimize both the power consumption and the minimum yield “equally”. As a result, we see that the EA-RESALLOC_MIXED algorithm does not perform as well as before. In fact, while the overall minimum yield is reduced the energy remains virtually the same. A possible reason is that the increase of λ does not have a sufficient impact on the energy component of our metric. In other words, the impact of the λ parameter on the minimum yield for our metric is not proportional to the impact on the energy consumption. As a result, λ has a much bigger impact on the minimum yield than on the energy consumption.

The case $\lambda = 0.5$ might be an inflexion point of our metric where our algorithm cannot reduce the energy enough to match the performance loss. As a result, such a value is likely not desirable and a value closer to 0 or to 1 is preferable.

Figure 16 plots results obtained $\lambda = 0.8$. In this case, we want to reduce energy, even if the performance loss is large. And indeed, the performance loss is significant. In fact, most minimum yield values are under 0.5. Unlike in Figure 15, however, the energy reduction is significant. We manage to achieve some consolidation and as a result we could decrease in some case the overall energy consumption by a small fraction. The performance loss still does not match the energy reduction by a large margin.

The problem here is that the k parameter of our metric is not related to both the minimum yield and the power consumption in the same way that λ is. This is why those results do not exhibit the performance that we would have expected with such a value of λ. Here for instance, we would want for example to use $k = 0.9$ to achieve the energy reduction wanted with $\lambda = 0.8$. Such relationship between the two parameters is not known at this time, and is left for future investigation.

Figures 17 and 18 plot respectively the average energy consumption and the average minimum yield for each algorithm, grouped by number of hosts in the instances. The best energy consumption is achieved by the GREEDY_MIXED_1 algorithm, as it focuses on the energy reduction. However, this algorithm also has the smallest minimum yield of all the algorithms. The opposite goes for the EA-RESALLOC_MIXED algorithm, which achieves the highest energy consumption and also the highest minimum yield. This demonstrates the fact that bi-criterion optimization is difficult,

Figure 19 plots the execution time of the algorithms with 64 hosts and 192 jobs. Here we can see that the EA-RESALLOC_MIXED algorithm computes an allocation in under a second. Of course, the GREEDY_MIXED_1 computes in virtual no time at all while compute the LPBOUND can take up to almost 20 seconds.

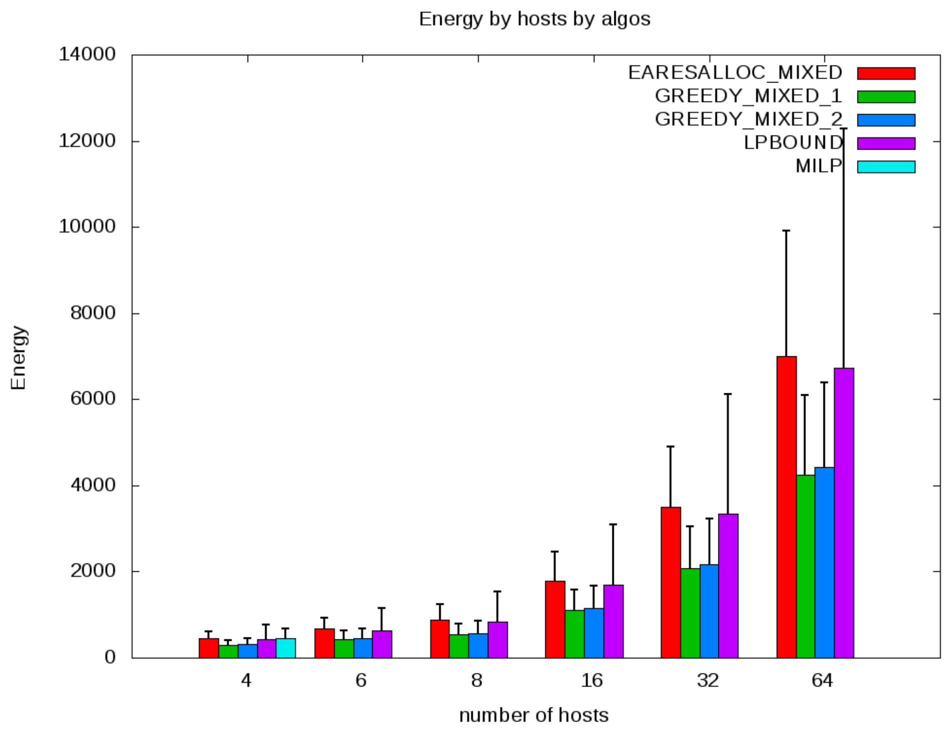

Figure 17: Energy consumed by each algorithms for each number of hosts

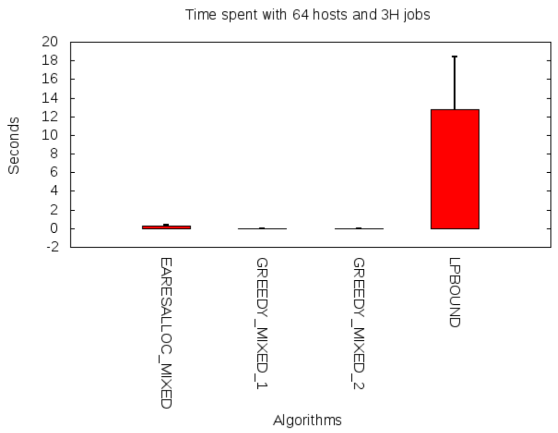

Figure 19: Execution time of the algorithms for 64 and 192 jobs

5. Discussion

In this section, we discuss some of the choices we have made in the previous section, as well as possible improvements to our approach.

The method to compute E^{max} described in Section 2.6 is only one possibility. Another method, for instance, would be to first sort the hosts by decreasing C^{max}, then to compute $E^{max} = \sum_i^{\lceil 2\sum_i(\alpha_i)\rceil}(C_i^{max})$. With this method, one could obtain an E^{max} value that is twice the achievable E^{max} (thus larger that the one computed in Section 2.6). This method, however, can avoid an effect that would occur in "stupid" allocations in which the energy part of Z vanishes. Furthermore, it computes the E^{max} value faster (which admittedly is not of great importance since the computation is done only once per instance).

For the MixedObjective problem, we have seen that the impact of λ is important, but not necessarily in the way that was initially intended. The reason, mentionned briefly earlier, is that choosing $\lambda = k$ for the EA-ResAlloc_Mixed algorithm may not the most relevant approach for adapting the algorithm in [12]. Studying results for various values of k, one can see that on most the $[0, 1]$ interval there is no effect on the energy consumption. Significant energy reduction are observed only on the $[0.7, 1]$ interval. Consequently, when solving our problems, the algorithm cannot achieve the balanced effect that could have been expected with $\lambda = 0.5$, and we observed energy consump-

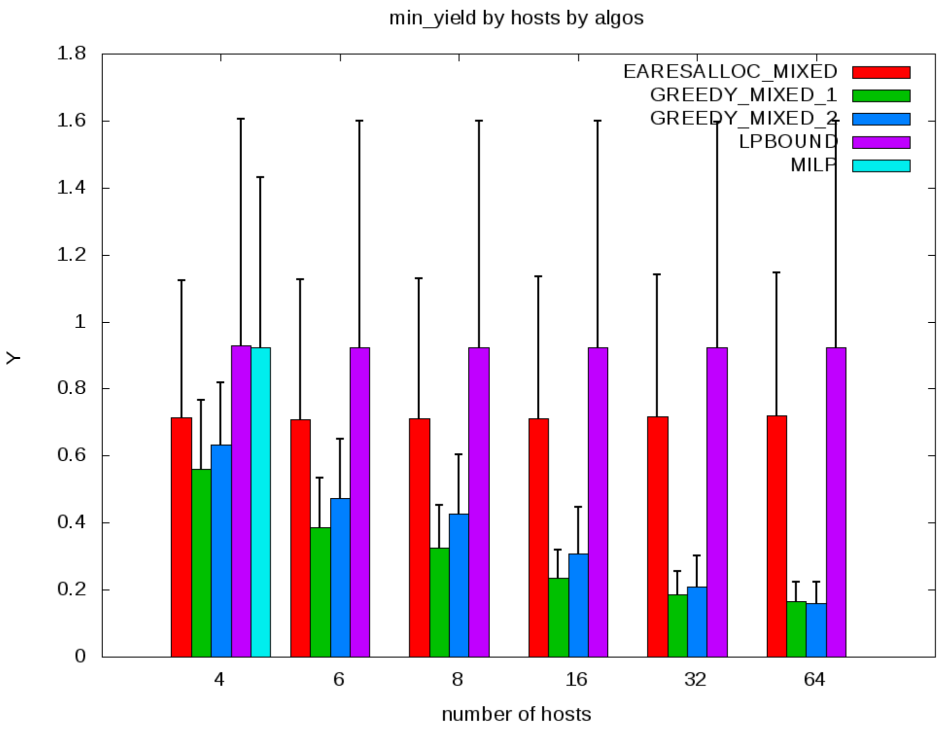

Figure 18: Minimum yield by each algorithms for each number of hosts

tion reduction only with $\lambda = 0.8$. In order to better balancing energy reduction with minimum yield maximization, one would likely need to define the energy-aware yield using a form close to the definition of Z, which is, after all, the optimization goal.

As seen in the results for the BoundedEnergy problem, the EA-ResAlloc_Bound_E algorithm fails as much as the greedy algorithms. A failure rate as high as 10% can be problematic (and recall that we considered at most $J = 3H$ jobs). The reason for these failures is that the algorithm leads to imbalanced consolidation of workloads (i.e., only a few hosts are overloaded). Improving the failure rate of EA-ResAlloc_Bound_E for such instances must thus come from more balanced consolidation. The energy-aware yield metric varies based on the current jobs allocated to a host. This means that, for example, a host already loaded will not have the same energy-aware yield than another host for the allocation of a particular job. While this is by design, one could imagine adding a balancing step to the algorithm, in order to produce a more balanced allocation, thus likely reducing the failure rate.

In this work, we do not consider the sum yield, which would be defined as follows using our notations:

$$\sum_i \sum_h \frac{\alpha_{ih}}{\alpha_i} .$$

Optimizing the sum yield would make sense in scenarios in which the throughput of the system is more of a concern than fairness. It would be straightforward to modify our problem formulation and algorithms to optimize the sum yield for the BoundedPower and BoundedYield problems. However, taking into account the sum yield for the MixedObjective problem is more challenging since a direct application of our techniques may lead to job starvation [16, 17]. Note that in [9] sum yield maximization is used as a second optimization step once the minimum yield has been maximized, so as to increase resource utilization.

Our approach takes into account both memory and CPU resources, and studies three different allocation problems. There are, however, several approaches and techniques for reducing energy consumption that we have not considered, such as Dynamic Voltage and Frequency Scaling (DVFS) and VM migration. With DVFS, processors can be configured to achieve particular trade-offs between compute speed and energy consumption. Migration can be used to adapt a resource allocation in the case of dynamic workloads. Both DVFS and migration thus afford more flexibility both for workload consolidation and for energy consumption reduction. Our approach could be augmented to account for DVFS and migration capabilities. Note that our linear program formulation would thus be larger (i.e., more constraints and variables), which may make it prohibitive to compute the optimal even for very small instances. Furthermore, the solution of the relaxed linear program could be further from the actual optimal.

6. Related Work

There is a large literature on the topic of resource allocation and resource management in clusters. Furthermore, with virtual machine (VM) technology it has become possible to easily consolidate and quickly adapt resource allocations. Consequently, many recent efforts have studied the resource allocation problem in light of these new capabilities. In this section we only discuss those works that consider for the power consumption of a resource allocation. We refer the reader to [9] for an extensive literature review of resource allocation techniques and algorithms that do not specifically take power consumption into account, and to [18] for a list of possible techniques for reducing energy in large scale distributed systems.

The simplest technique to reduce power consumption is to power down cluster nodes. Kamitsos et al. attempt to find an optimal policy for powering nodes up and down using a Markov decision process [19]. As in virtually all approaches, there is a tradeoff between performance and power consumption. Solving the Markov decision process makes it possible to find a Pareto-optimal tradeoff between these two metrics. A similar approach is taken in [20], in which 3 metrics are considered: queue waiting time, power consumption, and job blocking probability. In this work we also consider a multi-objective problem and power down nodes to conserve energy. The authors in [19] and [20] only consider jobs that need a whole host to run, whereas we allow the sharing of hosts by multiple jobs. Moreover, unlike in those other works, we do consider job memory requirements.

Beyond simply powering nodes up and down, a more fine-grain option is to use DVFS. Rong et al. [21] use DVFS for scheduling scientific applications on clusters. Leveraging load imbalance and communication delays of the applications, they demonstrate that power consumption can be reduced significantly with only marginal increases in application execution times. They mainly provide tools for programmers to achieve this energy reduction, whereas here we propose a system that manage the energy reduction itself. Etinski et al. [22] also use DVFS to exploit the load imbalance of large scale MPI applications. Essentially, they deliberately slow down shorter tasks to match the task with the longest execution time. They also take into account overclocking to reduce execution time on a small subset of the nodes, thus not increasing power consumption. They show that up to 60% CPU power consumption can be saved in imbalanced MPI applications. Although DVFS is not considered in this work, our problem formulation and our algorithms could easily be extend it to make use of it.

In this work we base our algorithms on a mixed integer linear program formulation of the resource allocation problem to decide where to place VM instances. A similar approach is used, for instance, by Petrucci et al. [23]. The linear program is solved periodically in a control loop fashion. Considering a heterogeneous cluster with DVFS-enabled nodes, a set of job placement constraints is derived and the objective is to minimize power consumption. This approach can also take advantage of VM instance migration, accounting to the corresponding overhead. Migration decisions are based on a notion of network cost, i.e., cost to transfer VM instance images over the network. The migration problem is likely far more complex because instances are often part of an application and interact with each others. Ideally, migration costs would account for the interactions between the VMs as well as for data locality. The work in [23] relies on a framework for monitoring and reconfiguring instances based on the actuators described in [24].

Current works that aims at reducing energy consumption using VM technology mainly focus on the dynamic workloads, with the objective of reducing migration costs while aggregating VM instances on a reduced set of hosts. Entropy [25] is a resource manager for homogeneous clusters that performs dynamic consolidation of resources based on constraint programming, using VM migration and taking its own overhead into account. Berral et al. [26] achieve significant power consumption reduction via resource consolidation using machine learning to make resource allocation decisions. Essentially, their approach favors the allocation of new jobs to already powered-up nodes, possibly using migration. Our approach is more service-oriented as we allocate directly all jobs instead of allocating jobs over time. Thus we do not consider job arrival times, but instead consider long-running services.

Hoyer et al. propose statistical allocation planning using two different approaches for resource allocation [27]. The first approach is pessimistic and allocates to each VM instance the maximum resource fraction it could need, using vector packing to perform the allocation. Their second approach is instead optimistic and tries to overbook each host while maintaining each job over a certain performance threshold. They mainly focus on dynamic tracking of resource needs of VM instances, and on how to react to changing needs, which is complementary to our allocation problem. Results indicate up to 27% savings in power compared to a resource allocation policy that always allocate to a job exactly the resources it need.

Power consumption reductions via intelligent resource allocation is not only studied at the cluster scale, but also at a global scale. For instance, in [28] Le et al. propose a framework to reduce costs in geographically distributed system. The goal is to exploit the differences and variabilities between the energy costs of data centers, the different time zone where these data centers are located, as well as their proximity to "green" power sources. The optimization problem is to minimize cost while meeting SLAs.

7. Conclusion

In this paper we have formulated the problem of energy-aware task allocation in commodity clusters as a multi-objective problem. We have reduced this problem to three single-objective problems: (i) optimize job performance given power consumption constraints; (ii) optimize power consumption given job performance constraints; and (iii) optimize a linear combination of job performance and power consumption. We have proposed five heuristics, four based on standard greedy algorithms and a novel heuristic called EA-ResAlloc. We have compared these heuristics to optimal and near-optimal solutions over ranges of problem instances using simulation. We have identified some heuristics that perform well in terms of energy savings while still leading to good job performance. These heuristics enforce consolidation on a subset of the cluster hosts judiciously chosen depending on the difficulty of the resource allocation problem. Finally, these heuristics have low computational cost and can thus be employed in practical settings. Overall, the proposed EA-ResAlloc is a promising heuristic that outperforms classical greedy algorithms in many relevant scenarios (in particular for the first two problems above). Although EA-ResAlloc can compute allocations quickly, in some cases (i.e., optimizing power consumption given job performance constraints for large problem instances with hundreds of hosts and jobs) one may favor the use of one of the greedy algorithms so as to compute the allocation in only a few seconds.

References

[1] U. E. P. Agency, Report to Congress on Server and Data Center Energy Efficiency, http://repositories.cdlib.org/lbnl/LBNL-363E/ (August 2007).

[2] W. Huang, J. Liu, B. Abali, D. K. Panda, A case for high performance computing with virtual machines, in: Proc. of the 20th Annual Intl. Conf. on Supercomputing, 2006, pp. 125–134.

[3] B. Urgaonkar, P. Shenoy, T. Roscoe, Resource Overbooking and Application Profiling in Shared Hosting Platforms, SIGOPS Oper. Syst. Rev. 36 (SI) (2002) 239–254.

[4] K. Shen, H. Tang, T. Yang, L. Chu, Integrated Resource Management for Cluster-based Internet Services, in: Proc. of the 5th Symposium on Operating Systems Design and Implementation, 2002.

[5] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, T. Wood, Agile dynamic provisioning of multi-tier Internet applications, ACM Trans. Auton. Adapt. Syst. 3 (1) (2008) 1–39.

[6] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, A. Kemper, An integrated approach to resource pool management: Policies, efficiency and quality metrics, in: Proc. of the IEEE Intnl. Conf. on Dependable Systems and Networks, 2008, pp. 326–335.

[7] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, A. Tantawi, Dynamic Placement for Clustered Web Applications, in: Proceedings of the 15th International Conference on the World Wide Web, 2006, pp. 595–604.

[8] D. Carrera, M. Steinder, I. Whalley, J. Torres, E. Ayguadé, Utility-based placement of dynamic web applications with fairness goals, in: IEEE Network Operations and Management Symposium, 2008, pp. 9–16.

[9] M. Stillwell, D. Schanzenbach, F. Vivien, H. Casanova, Resource allocation algorithms for virtualized service hosting platforms, Journal of Parallel and Distributed ComputingIn press (Digital Object Identifier: 10.1016/j.jpdc.2010.05.006).

[10] X. Fan, W.-D. Weber, L. A. Barroso, Power provisioning for a warehouse-sized computer, in: Proceedings of the 34th annual international symposium on Computer architecture, ISCA '07, ACM, 2007, pp. 13–23.

[11] M. Stillwell, F. Vivien, H. Casanova, Dynamic Fractional Resource Scheduling for HPC Workloads, in: Proc. of the 24th IEEE Intl. Parallel & Distributed Processing Symp., 2010.

[12] D. Borgetto, G. Da Costa, J.-M. Pierson, A. Sayah, Energy-Aware Resource Allocation, in: Proc. of the Energy Efficient Grids, Clouds and Clusters Workshop (E2GC2), IEEE, 2009, p. (electronic medium).

[13] Gnu linear programming kit.
URL `http://www.gnu.org/software/glpk/`

[14] IBM ILOG CPLEX (2010).
URL `http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/`

[15] Grid 5000.
URL `https://www.grid5000.fr/`

[16] M. A. Bender, S. Chakrabarti, S. Muthukrishnan, Flow and Stretch Metrics for Scheduling Continuous Job Streams, in: Proc. of the Symp. On Discrete Algorithms, 1998, pp. 270–279.

[17] A. Legrand, A. Su, F. Vivien, Minimizing the Stretch when Scheduling Flows of Divisible Requests, J. of Scheduling 11 (5) (2008) 381–404.

[18] D. Careglio, G. Da Costa, R. Kat, A. Mendelson, J.-M. Pierson, Y. Sazeides, Hardware leverages for energy reduction in large scale distributed systems, Tech. rep. (2010).

[19] Y. Kamitsos, L. L. H. Andrew, H. Kim, M. Chiang, Optimal Sleep Patterns for Serving Delay Tolerant Jobs, in: ACM eEnergy, University of Passau, Germany, 2010.
URL `http://netlab.caltech.edu/lachlan/abstract/eEnergySleep.pdf`

[20] D. Niyato, S. Chaisiri, L. B. Sung, Optimal power management for server farm to support green computing, in: CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, IEEE Computer Society, Washington, DC, USA, 2009, pp. 84–91. doi:http://dx.doi.org/10.1109/CCGRID.2009.89.

[21] R. Ge, X. Feng, K. W. Cameron, Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters, in: SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing, IEEE Computer Society, Washington, DC, USA, 2005, p. 34. doi:http://dx.doi.org/10.1109/SC.2005.57.

[22] M. Etinski, J. Corbalan, J. Labarta, M. Valero, A. Veidenbaum, Power-aware load balancing of large scale mpi applications, in: IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing, IEEE Computer Society, Washington, DC, USA, 2009, pp. 1–8. doi:http://dx.doi.org/10.1109/IPDPS.2009.5160973.

[23] V. Petrucci, O. Loques, D. Mossé, A Dynamic Optimization Model for Power and Performance Management of Virtualized Clusters, in: ACM eEnergy, University of Passau, Germany, 2010.

[24] V. Petrucci, O. Loques, D. Mossé, A framework for dynamic adaptation of power-aware server clusters, in: SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing, ACM, New York, NY, USA, 2009, pp. 1034–1039. doi:http://doi.acm.org/10.1145/1529282.1529509.

[25] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, J. Lawall, Entropy: a Consolidation Manager for Clusters, Research Report RR-6639, INRIA (2008).
URL `http://hal.inria.fr/inria-00320204/en/`

[26] J. L. Berral, Íñigo Goiri, R. Nou, F. Julià, J. Guitart, R. Gavaldà, J. Torres, Towards energy-aware scheduling in data centers using machine learning, in: ACM eEnergy, University of Passau, Germany, 2010.

[27] M. Hoyer, K. Schröder, W. Nebel, Statistical static capacity management in virtualized data centers supporting fine grained QoS specification, in: ACM eEnergy, University of Passau, Germany, 2010.

[28] K. Le, R. Bianchini, M. Martonosi, T. D. Nguyen, Cost- and energy-aware load distribution across data centers, in: HotPower'09 : Workshop on Power Aware Computing Systems, 2009.