Heuristic Resource Allocation Algorithms for Maximizing Allowable Workload in Dynamic, Distributed Real-Time Systems

David Juedes, Frank Drews, Lonnie Welch, and David Fleeman
Center for Intelligent, Distributed, and Dependable Systems
School of Electrical Engineering and Computer Science
Ohio University
Athens, Ohio 45701 U.S.A.
{juedes,drews,welch,fleeman}@ohio.edu

Abstract

This paper examines several heuristic algorithms for the *maximum allowable workload* (MAW) problem for real-time systems with tasks having variable workloads. Briefly, the problem concerns the allocation of n tasks to m processors, where each task t is characterized by a function $t.r(w)$ that gives the running time of the task in terms of its workload (or input size) w. The objective of the *maximum allowable workload* problem is to find an allocation of tasks to processors so that the allocation is feasible (no task misses its deadline) when each task is given a workload of w or smaller and w is maximized. This optimization problem uses a robustness measure that is closely related to the MAIL (maximum allowable increase in load) metric recently proposed by Gertphol et. al. The main contribution of this paper is the the comparison of several heuristic algorithms for the MAW-RMS problem. Hillclimbing, random search, simulated annealing, and first-fit heuristics are presented and evaluated via simulation. As we show here, the first-fit greedy heuristic produces solutions of a reasonable quality compared to the other algorithms. In addition, we demonstrate the applicability of our model in air defense systems.

1 Introduction

The use of distributed real-time and embedded applications as used in such critical areas as air defense, robotic control, and satellite constellations gives rise to new problems concerning the management of resources in such systems. These systems are generally characterized by an unpredictable behavior of the environments that prohibits the specification of meaningful arrival rates of events driving transient computations. At the same time, the periodic computations may be required to process data of varying sizes. Moreover, the size of the data to be processed in an arbitrary period sometimes cannot be determined at the time of system engineering, nor can a meaningful upper bound be found for the size of the data. Thus, a useful worst-case execution time cannot be obtained for such periodic processes *a priori*. Such systems with their real-time constraints are best described as *dynamic real-time systems*.

Traditional methods for achieving real-time performance do not always work well in dynamic environments because they have overwhelmingly concentrated on periodic tasks with *a priori* workloads. In early work, Abdelzaher and Shin [2, 3] examined a branch and bound algorithm for off-line scheduling of communicating tasks and messages with known arrival rates and resource requirements, where the optimization objective was the minimization of the maximum task lateness. Peng, Shin, and Abdelzaher [17] gave another branch-and-bound algorithm for allocating communicating periodic tasks with the objective of minimizing the maximum normalized system response time, called the system hazard. The Q-RAM project uses a similar objective function for optimizing resource allocations; however, it works for utility-based soft real-time applications [18, 19, 13, 12]. Other authors presented algorithmic approaches for assigning real-time tasks based on general purpose local search heuristics. Coli and Palazzari [6], DiNatale and Stankovic [7], Nicholson [15], and Tindell *et al.* [22] gave algorithms based on simulated annealing to optimally allocate periodic tasks

to resources. Algorithms based on genetic algorithms were proposed by Baccouche [5], Greenwood *et al.* [10], and Sandnes [20].

In contrast to much of the aforementioned earlier work on resource allocation in real-time systems, the research described here does not assume an upper bound on the size of the input data or an upper bound on the worst-case execution time for each task. Instead, it assumes the existence, for each task T_i, of an (arbitrary) function $T_i.r(w)$ that gives an upper bound on the running time of task T_i on inputs of size w. As described in detail below, our objective is to produce an allocation of resources to tasks that remains feasible for the largest possible system workload w. This is the *maximum allowable workload* (MAW) problem, and it has several variants based on the underlying process scheduling algorithm. MAW-RMS is the version that uses fixed priority rate monotonic scheduling. MAW-EDF is the version that uses the earliest deadline first scheduling approach.

Maximizing the allowable workload produces resource allocations that are robust with respect to increases in workload. Robustness metrics have been a source of recent work [9, 4]. In particular, the recent work by Gertphol *et al.* [9] presents a robustness metric that is very close to maximum allowable workload. They define a performance metric "MAIL," which stands for "maximum allowable increase in load" and present several heuristic algorithms for finding resource allocations that allow a maximum increase in load. Briefly, they attempt to produce an allocation of resources to tasks so that the system can absorb the largest increase in load level α in the sense that the allocation of resources remains feasible when the "load level" of each task is increased by a factor of $(1+\alpha)$. The *MAIL* metric can be seen as roughly equivalent to maximum allowable workload if we measure the running-times of tasks in terms of a single load increase parameter α. Although their work is based on a more general system model that allows dependency relationships, the model used for this paper allows us to prove theoretical results for the described algorithms.

1.1 The Optimization Problem

Here we use the following model for real-time systems. Each real-time system consists of a collection n processors $P = \{P_1, \ldots, P_n\}$ and a collection of m independent periodic tasks $T = \{T_1, \ldots, T_m\}$. Each task T_i has a period $T_i.p$ and a deadline $T_i.d = T_i.p$. Furthermore, each task T_i has a collection of functions $T_i.r_j(w)$ that describe the running-times of task T_i on each processor P_j with workload w. It is assumed that $T_i.r_j(w)$ is a monotonically non-decreasing function in w.

For shorthand, we describe a real-time system by the tuple $\langle T, P\rangle$. The maximum allowable workload problem is formalized as follows.

Optimization Problem: (Maximum Allowable Workload) MAW-RMS/MAW-EDF
Input: $\langle T, P\rangle$
Output: An allocation of tasks to processors, $alloc : T \longrightarrow P$ and a workload w such that the allocation *alloc* is *feasible* whenever all tasks do not exceed the workload w. Here, we say that an allocation is feasible if all tasks on a single processor meet their deadlines when scheduled using RMS [14]. The optimization problem MAW-EDF can be defined by replacing RMS with EDF in the above definition.
Objective Function: Maximize w.

Note that the optimum solution for MAW-RMS or MAW-EDF has the property that the allocation is feasible with workload w but not feasible with workload $w + 1$. Also note that it is easy to see that the optimization problem MAW-EDF is NP-hard since it can be used to solve bin-packing [8]. To see this, consider an instance of bin-packing that asks whether a collection of

items $\{i_1, \ldots, i_n\}$ of sizes (whose values are rationals) $s_1 = n_1/d_1, s_2 = n_2/d_2, \ldots, s_n = n_n/d_n$ can be packed into K bins. This instance can be transformed into an instance of MAW-EDF with n tasks and K processors. Define a running-time function $T_i.r(1) = 0$ and $T_i.r(j) = n_i$ for all $j \geq 2$, and let $T_i.p = d_i$. Then, the optimum solution for this instance of MAW-EDF can feasibly handle a workload of 2 if only if the n items $i_1, \ldots, i_n$ can be packed into K bins. It follows that MAW-EDF is NP-hard. Therefore, it seems likely that MAW-RMS is also NP-hard. Hence, it is likely that we will not be able to find an efficient exact algorithm for it.

In our previous work (cite ipdps2004 paper) we proposed a first fit approach for MAW-RMS and analyzed its performance when (i) all tasks are independent and periodic and (ii) when the running times of tasks in the system are well-behaved. For an air defense system it was shown that it has these properties. In addition, we showed analytically that FF is guaranteed to produce a solution that is at least 41% of optimal, asymptotically, when the running-times of all tasks are input dependent and well-behaved. Moreover, we show that if at most 12% of the system utilization is consumed by input independent tasks (e.g., constant time tasks), then FF is guaranteed to produce a solution that is at least 33% of optimal, asymptotically. Experimental results also indicate that FF may perform much better in practice. This suggests that FF may be a reasonable approach for resource allocation in real-time systems with variable workloads.

In this paper we introduce and examine several heuristic algorithms and compare them with FF.

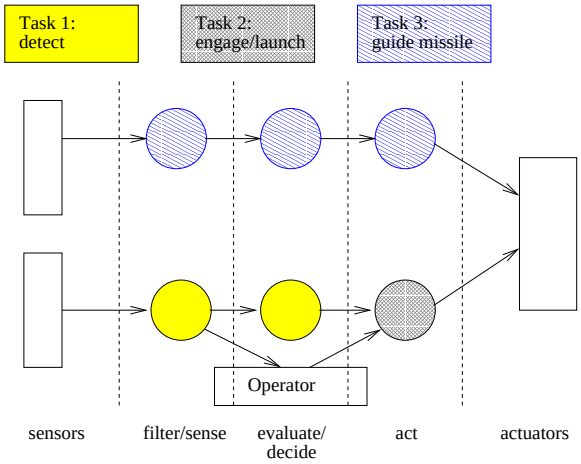

Figure 1: A Motiving Example From DynBench [23]

1.2 A Motivating Example

The notion of tasks which have workload-dependent execution times originated from the study of the generic air defense system [23] depicted in the Figure 1. The *detect* task identifies threats to a defended entity. The task runs periodically and performs the functions of filtering and evaluating radar tracks. When a threat is detected, the *detect* task triggers the *engage* task, which fires a missile at the threat. After a missile is in flight, the *guide missile* task keeps the missile on the correct course. The *guide missile* task executes periodically, uses sensor data to track the threat, recalculates the flight path for the missile, and issues guidance commands to the missile.

All three of these tasks have resource needs that are environment-dependent. The execution time of the *detect* task is primarily workload dependent. Since the task evaluates each radar track

to determine if it is a potential threat, its execution time is a function of the number of radar tracks in the environment. The workload of the *engage* task is also variable since it is activated by events which occur at rates that are determined by the external environment. Similarly, the work performed by the *guide missile* depends on the number of missiles in flight.

Unlike traditional approaches to real-time computing, which characterize the resource needs of a task by a worst case execution time (WCET) [14], we characterize the resource needs of these tasks by execution profile functions. These functions compute the resource need as a function of workload. The execution profile of the *detect* task [21] has a resource requirement of $f(r) = 0.0869r^2 + 15.4374r + 614.8615$ microseconds, where "r" is the number of radar tracks. In the terminology used in this paper, the execution profile of the detect task has a well-behaved, input dependent component $f'(r) = 0.0869r^2 + 15.4374r$ and a input independent component $f^*(r) = 614.8615$. In contrast, the workload of the *engage* task for each activation is a function of the number missiles m in flight. The execution profile of the *engage* task is $g(m) = 12897m + 45610$ microseconds. Finally, the *guide missile* task has an execution profile that depends both on the radar tracks r and the number of missiles in flight m. The execution profile of the guide missile track is $h(r, m) = 0.0869r^2 + 15.4374r + 12903.909m + 46475.609$ microseconds. These profile functions are obtained by analyzing execution profiles at various workloads and then fitting data to the appropriate curve [24].

We note that concurrency is often used in order to meet the real-time constraints of the tasks. Thus, during operation, there may be many replicas of the three tasks in the air defense system. When the number of radar tracks grows too large for a single replica of the *detect* task to process all tracks within the required time bound, one or more replicas are created and the radar tracks are partitioned among them. Additionally, pipeline concurrency is obtained by replicating the subtasks of the *detect* task. In a similar manner, the *guide missile* task is replicated as necessary to meet deadlines, and its subtasks are distributed. Replication is also used for the *engage* task when heavy workloads are anticipated. Thus, an important problem to solve for this system is how to allocate the tasks and subtasks in a manner that allows real-time constraints to be met and that minimizes the need for reallocations (which create overhead in the system).

1.3 Contribution and Organization of the Paper

In the next section, we present a collection of heuristic algorithms for the MAW-RMS problem. class of algorithmic techniques. In particular, we present heuristics for resource allocation based on hillclimbing, random search, and simulated annealing. In section 3, we compare the performance of these algorithmic approaches on large collections tests.

2 Heuristic Algorithms for MAW-RMS

In this section, we examine a collection of sub-optimal algorithms to solve the general version of MAW-RMS, i.e., we assume that the running time of each task may depend on the machine on which it is being executed. We give four algorithms to solve the general version of MAW-RMS based on first fit, random search, simulated annealing, and hillclimbing techniques.

We first describe our first fit approach to MAW-RMS (Algorithm 2) in more detail. The algorithm employs an algorithm by Oh and Baker that we present as Algorithm 1. Given a workload w for each task in the system, Algorithm 2 tests to see if the Algorithm 1 will find an allocation of tasks to processors that will satisfy all the real-time constraints with this workload. Algorithm 1 tests to see if the first-fit allocation of tasks to processors via the Liu and Layland $m(2^{1/m} - 1)$ utilization criteria uses the required number of processors.

Algorithm 1 Greedy First Fit Algorithm by Oh and Baker [16]

Input: $\langle T, P \rangle$ and workload w;
Output: An allocation $alloc : T \longrightarrow P$ and "Feasible" or "Not Feasible"
for each job i **do**
  set $j = 1$; n= # of processors in P;
  while job i has not been allocated and $j \leq n$ **do**
    set $x = |\{T_k | alloc(T_k) = P_j\}| + 1$;
    if $\left(\sum_{alloc(T_k)=P_j} \frac{T_k.r(w)}{T_k.p} \right) + \frac{T_i.r(w)}{T_i.p} \leq x(2^{\frac{1}{x}} - 1)$ **then**
      set $alloc(T_i) = P_j$;
    else
      set $j = j + 1$;
    end if
  end while
  if $j > n$ **then**
    return "Not Feasible";
  end if
end for
return "Feasible";

When given a real-time system $\langle T, P \rangle$ and a workload w, Algorithm 1 returns whether this workload can be feasibly scheduled on the n processors. If it can be feasibly scheduled, then the algorithm returns that appropriate allocation of resources ($alloc$) that enables us to execute the tasks using this workload. The algorithm we describe as Algorithm 2 uses binary search to find a w such that Algorithm 1 determines that the real-time system with workload w is feasible, but the real-time system with workload $w + 1$ is not feasible. Algorithm 2 produces the appropriate allocation of resources as a result. We refer to Algorithm 2 as First-Fit (FF).

The heuristics presented in this section make use of the following basic approach: first find an allocation of tasks to resources $alloc : T \longrightarrow P$, and then find the maximum workload w that can be used. Once an allocation of tasks to resources is fixed, the maximum feasible workload can be found quickly using Algorithms 3 and 4. All heuristic approaches use these algorithms to find the largest feasible workload for a given allocation. The three algorithms differ in the way that the space of possible resource allocations is searched.

Our next approach to search the space of possible allocations is a random search algorithm that searches through a fixed number of random allocations. Random search is given as Algorithm 5 and we refer to the algorithm as RS. In our simulation results in the next section, we test random search with 100000 and 400000 iterations.

Now we propose an approach based on the local search procedure simulated annealing [11]. We give our approach as Algorithm 6. The simulated annealing algorithm is a randomized local search procedure that attempts to avoid getting stuck in local minima (or maxima) by accepting negative changes to the solution, i.e., the new allocation may not be able to handle as much workload. The simulated annealing process starts at a high temperature T_0 with an initial solution. The solution is then changed randomly. If the change improves the solution, then the new solution replaces the old one. If the change makes the solution worse by Δ, the change is accepted with probability $e^{-\frac{\Delta}{T_0}}$. In general, large negative changes may be accepted at high temperatures, but only small negative changes are accepted at low temperatures.

Algorithm 2 First Fit Approximation Algorithm for MAW-RMS

Input: $\langle T, P \rangle$.
Output: An allocation $alloc : T \longrightarrow P$ and a workload w
Comment: *Uses binary search to find a w such that Algorithm 1 returns "Feasible" for w and "Not Feasible" for $w + 1$.*
set $w = 1$;
while Algorithm 1 on input (T, P, w) returns "Feasible" **do**
  set $w = w * 2$;
end while
set $low = w/2$; set $high = w$;
while $high - low > 1$ **do**
  set $w = \frac{low+high}{2}$;
  if Algorithm 1 on input (T, P, w) returns "Feasible" **then**
    set $low = w$;
  else
    set $high = w$;
  end if
end while
return $alloc$ and $w = low$

Algorithm 3 Using RMA to Check Feasibility of an Allocation/Workload

Input: $\langle T, P \rangle$, $alloc : T \longrightarrow P$ and w.
Output: Feasible or Not Feasible
for $j = 1$ to n **do**
  set $l = |\{T_k | alloc(T_k) = P_j\}|$;
  if $\sum_{alloc(T_i)=P_j} \frac{T_i.r_j(w)}{T_i.p} > l(2^{\frac{1}{l}} - 1)$ **then**
    return "Not Feasible";
  end if
end for
return "Feasible";

Our simulated annealing algorithm has four basic parameters that affect the performance of the algorithm: the initial temperature, the stop temperature, the cooling constant c, and the iteration constant m. For our experiments, we used an initial temperature T_0 of 50 degrees and a stop temperature of 1 degree. The iteration constant m was set to 2100 and 8400, and hence the inner loop generated 2100 and 8400 new allocations. The function F generates new allocations by randomly picking one task T_i and one processor P_j, and moving task T_i to processor P_j. After m iterations, the current temperature T_0 is cooled based on a geometric cooling schedule

$$T_0 = T_0 \cdot c, \quad c \in (0, 1),$$

where c is the (constant) cooling rate, which was set to $c = 0.9$ [1]. Hence, the simulation performs approximately 100000 iterations and 400000 iterations, respectively.

Another crucial variable in the simulated annealing algorithm is the choice of an initial allocation. In our simulations, we tested three different versions of Algorithm 6: one that used an initial allocation of all of the tasks to a single processor (O allocation), one that uses a random initial allocation (R allocation), and one that used the first fit algoritmn 2. These are referred to as SA(O), SA(R), and SA(FF), respectively, in the next section.

Algorithm 4 Generic Algorithm that Finds the Optimal Workload for an Allocation

Input: $\langle T, P \rangle$ and an allocation $alloc : T \longrightarrow P$
Output: w such that the allocation with workload w is feasible, but the allocation with workload $w + 1$ is not.
set $w = 1$;
repeat
  $w = w * 2$;
  call Algorithm 3 to see if alloc is feasible with workload = w;
until alloc is not feasible with workload w
set $low = 1$; set $high = w$;
while $(high - low > 1)$ **do**
  set $w = \frac{high+low}{2}$;
  call Algorithm 3 to see if alloc is feasible with workload = w;
  if (feasible) **then**
    set $low = w$;
  else
    set $high = w$;
  end if
end while
return low;

Algorithm 5 Random Search Algorithm for MAW-RMS

Input: $\langle T, P \rangle$ and # of iterations
Output: An allocation $alloc : T \longrightarrow P$ and w
set $max = 0$;
for $i = 1$ to # of iterations **do**
  for $j = 1$ to m **do**
    pick x from $\{1, 2, \ldots, n\}$ at random using the uniform distribution.
    set $alloc(T_j) = P_x$;
  end for
  call Algorithm 4 to find maximum workload w for allocation $alloc$.
  if $(w > max)$ **then**
    set $max = w$; set $alloc' = alloc$;
  end if
end for
return $alloc'$, max;

Algorithm 6 Simulated Annealing Approach to MAW-RMS

Input: $\langle T, P \rangle$, $Initial\_Temperature$, $Stop\_Temperature$, c, m
Output: An allocation $alloc : T \longrightarrow P$ and workload w.
set $T_0 = Initial\_Temperature$;
set $T_1 = Stop\_Temperature$;
set $alloc = Initial\_Resource\_Allocation(T, P)$;
call Algorithm 4 to find the maximum workload w for $alloc$.
while $T_0 > T_1$ **do**
  for $j = 1$ to m **do**
    set $alloc' = F(alloc)$; {Generate a new allocation from $alloc$}
    call Algorithm 4 to find the maximum workload w' for $alloc'$.
    set $diff = w' - w$;
    if $diff > 0$ **then**
      set $w = w'$; set $alloc = alloc'$;
    else
      set x =random value from $[0, 1]$;
      if $(x < e^{-\frac{diff}{T_0}})$ **then**
        set $w = w'$; set $alloc = alloc'$;
      end if
    end if
  end for
  set $T_0 = c * T_0$; { Decrement T_0 }
end while
output $alloc$, w;

Algorithm 7 Hillclimbing Approach for MAW-RMS

Input: $\langle T, P \rangle$;
Output: An allocation $alloc : T \longrightarrow P$ and workload w.
set $alloc = Initial\_Allocation(T, P)$;
call Algorithm 4 to find the maximum workload w for $alloc$;
set $done = false$;
while (not done) **do**
  set $done = true$;
  for each neighbor $alloc'$ of $alloc$ **do**
    call Algorithm 4 to find the maximum feasible workload w' for $alloc'$;
    if $(w' > w)$ **then**
      set $done = false$; set $alloc = alloc'$; set $w = w'$;
    end if
  end for
end while
return $alloc$ and w;

Our final approach to search the space of all resource allocations is based on the hillclimbing technique and is given as Algorithm 7. This algorithm starts with an initial allocation of tasks to processors, and then performs a local search on the space of all allocations from that starting point. For a given allocation of tasks to processors, the local search algorithm examines the collection of all *neighbors* of the allocation. In our algorithm, a neighbor consists of an allocation that differs from the original allocation on at most one task. Since there are n processors and m allocations, it follows that there are exactly $(n-1) \cdot m$ neighbors of any allocation. The algorithm examines the maximum workload possible for each allocation in the neighborhood, and picks the allocation that produces the largest value as the next choice of an allocation. This process continues until the algorithm can no longer improve the value of the solution. Since it is possible for hillclimbing to become stuck in a local maximum, the choice of an appropriate starting allocation is crucial. Here we use one version of Algorithm 7 which is given a random initial allocation. We refer to this version of hill climbing als HC(R).

2.1 Experimental Setup

To test the approaches described in sections 4 and 5, we performed two collections of tests. In the first collection of tests, we considered only instances of the MAW-RMS problem with input dependent tasks. The test instances consisted of groups of $m \in \{20, 40, \ldots, 100\}$ tasks running on 10 processors. Each processor was described by its profile in terms of its speed factor. The values of the speed factors for each processor were chosen from the range $[10, 30]$ at random. Each periodic task was given a period in the range $[2500, 5000]$ milliseconds, selected uniformly at random. The running-time function (in terms of the number of milliseconds) for each task was determined by generating a random pseudo-polynomial with coefficients chosen uniformly at random from the range $[0, 100]$. The random pseudo polynomials were generated as follows. All terms in the pseudo polynomials came from the set $TP = \{x^2 \log x, x^2, x \log x, x\}$. The pseudo polynomials were chosen at random so that the probability of the largest term in the pseudo-polynomial was determined by the following table.

Largest Term	Probability	Largest Term	Probability
x	$\frac{1}{2}$	$x \log x$	$\frac{1}{4}$
x^2	$\frac{1}{8}$	$x^2 \log x$	$\frac{1}{8}$

In each pseudo polynomial, each lower order term was included with probability equal to one-half. This approach produced low degree pseudo polynomials with high probability. The running time profile of each Task is related to a profile host with speed factor 1. Consequently, the running time of a given task was translated between machines by dividing the running time of the task in accordance with its running time profile by the speed factor of the assigned machine.

2.2 Experimental Results

Figure 2 shows the performance of our algorithms on the first collection of test data. Simulated annealing and random search tend to outperform first fit when the number of applications is small. By increasing the number of iterations performed by simulated annealing and random search from 100000 to 400000, both algorithms found better solutions for the 20 and 40 application scenarios. However, since the number of iterations of simulated annealing and random search for scenarios with more than 60 applications is not increased further, first fit eventually outperforms all the heuristics presented in this paper, except SA(FF). Hill climbing turns out to be the worst allocation algorithm regardless of the scenario.

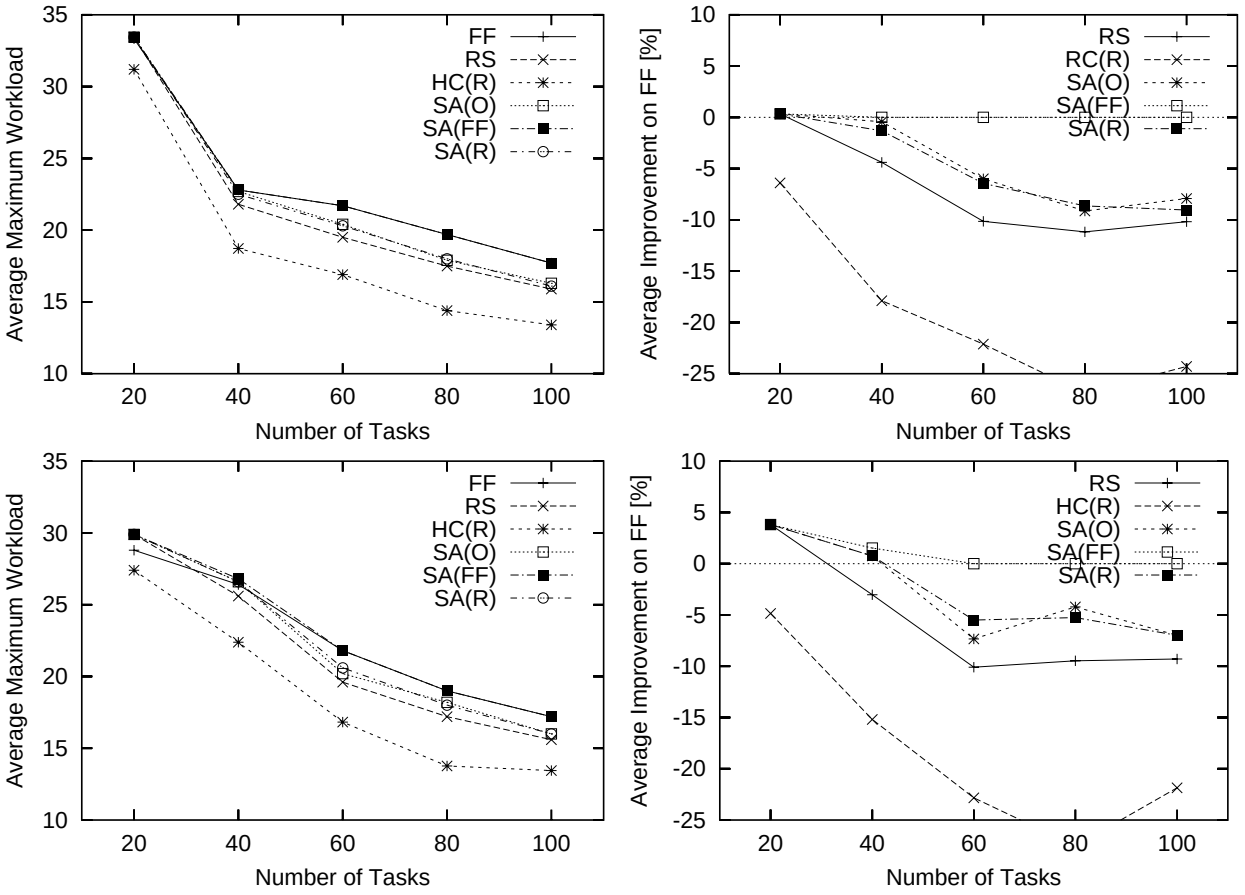

Figure 2: Performance on the first collection of tests. The results for simulated annealing and random search are based on 100000 iterations (above) and 400000 iterations (below). The left diagrams show the average maximum workload obtained. The diagrams on the right show the average percental improvement of workload on first fit.

While most of the algorithms produce reasonable solutions, their total running times were significantly different. Tables 3 and 2 provide the running times of the algorithms on the second set of test instances for 100000 iterations and 400000 iterations, respectively. All simulations were performed on a 440Mhz Sun Ultra 10 workstation

The second collection of experiments is basically equivalent to our first collection except that we considered both input dependent and input independent (constant time) tasks instances. The percentage of input dependent tasks was set to 85% and the percentage of input independent tasks to 15%. The input independent running times were chosen randomly from the range $[1500, 2000]$.

In figure 3, we show the performance of our algorithms on the second collection of test data. The results are very similar to the results produced by the first collection. Simulated annealing and random search find better solutions than first fit for small numbers of applications, and hill climbing always produces the worst allocation. The major difference in the results occur when simulated annealing is applied to the result produced by first fit. When there are workload independent applications, simulated annealing applied to the first fit solution tends to be more capable of finding a better allocation than when there are only workload dependent applications. The running times of the algorithms on the second collection of tests are essentially the same as on the first collection of test data.

# of Tasks	FF	RS	SA(O)	SA(R)	SA(FF)	HC(R)
20	0.01	155.47	164.92	164.43	165.03	0.86
40	0.03	266.05	272.9	272.95	277.04	2.47
60	0.06	327.47	336.93	336.93	336.37	5.15
80	0.12	390.71	407.42	404.86	400.83	7.77
100	0.19	480.83	492.08	498.12	488.41	10.31

Table 1: Running time of the algorithms for the first test scenario with 100000 iterations.

# of Tasks	FF	RS	SA(O)	SA(R)	SA(FF)	HC(R)
20	0.01	635.36	680.48	679.48	660.96	0.75
40	0.03	1047.78	1101.62	1099.14	1097.26	3.14
60	0.06	1278.93	1315.54	1312.32	1309.21	5.52
80	0.12	1570.34	1615.47	1613.80	1642.46	7.71
100	0.19	1870.29	1917.39	1944.54	1913.38	7.41

Table 2: Running time of the algorithms for the first test scenario with 400000 iterations.

In addition the three heuristics presented here, two other heuristics were implemented and tested: a worst-fit-approximation algorithm and a best-fit-approximation algorithm. It turned out, that the best-fit-approximation algorithm performs much the same than the first-fit approximation algorithm. The worst-fit-approximation algorithm performed on average about 10% to 15% worse than first-fit and next-fit.

2.3 A Motivating Example Revisited

In this subsection, we examine the performance of the aforementioned heuristics on example data gathered from the motivating example of a generic air defense system given in section 1.3. Recall that our example contained three basic types of tasks: the *detect* task, the *engage* task, and the *guide missile* task. As mentioned earlier, the profile functions for these three basic types of tasks depend on r, the number of radar tracks, and m, the number of missiles in flight. The profile function for the engage task was $g(m) = 12987m + 45610$ microseconds. The profile function for the detect task was $f(r) = 0.0869r^2 + 15.4374r + 614.8615$ microseconds. Finally, the profile function for the guide missile task is $h(r, m) = 0.0869r^2 + 15.4374r + 12903.909m + 46474.609$ microseconds. To match the parameters of our approach, we will assume that $r = m$, and hence that $g(m) = g(r) = 12987r+45610$, and $h(r, m) = h(r) = 0.0869r^2+15.4374r+12919.3464r+46474.609$.

Here we examine an example real-time system where these three basic tasks are replicated. For comparison, we consider three possible scenarios on a system with 20 processors: first, when we have 20 detect tasks, 5 engage tasks, and 10 guide missile tasks; second, when we have 15 detect tasks, 2 engage tasks, and 5 guide missile tasks; and, third, when we have 30 detect tasks, 10

Scenario	FF	RS	SA(FF)	HC(R)
I	180	175	190	162
II	210	198	217	187
III	128	119	131	118

Table 3: Maximum number of radar tracks for the three scenarios.

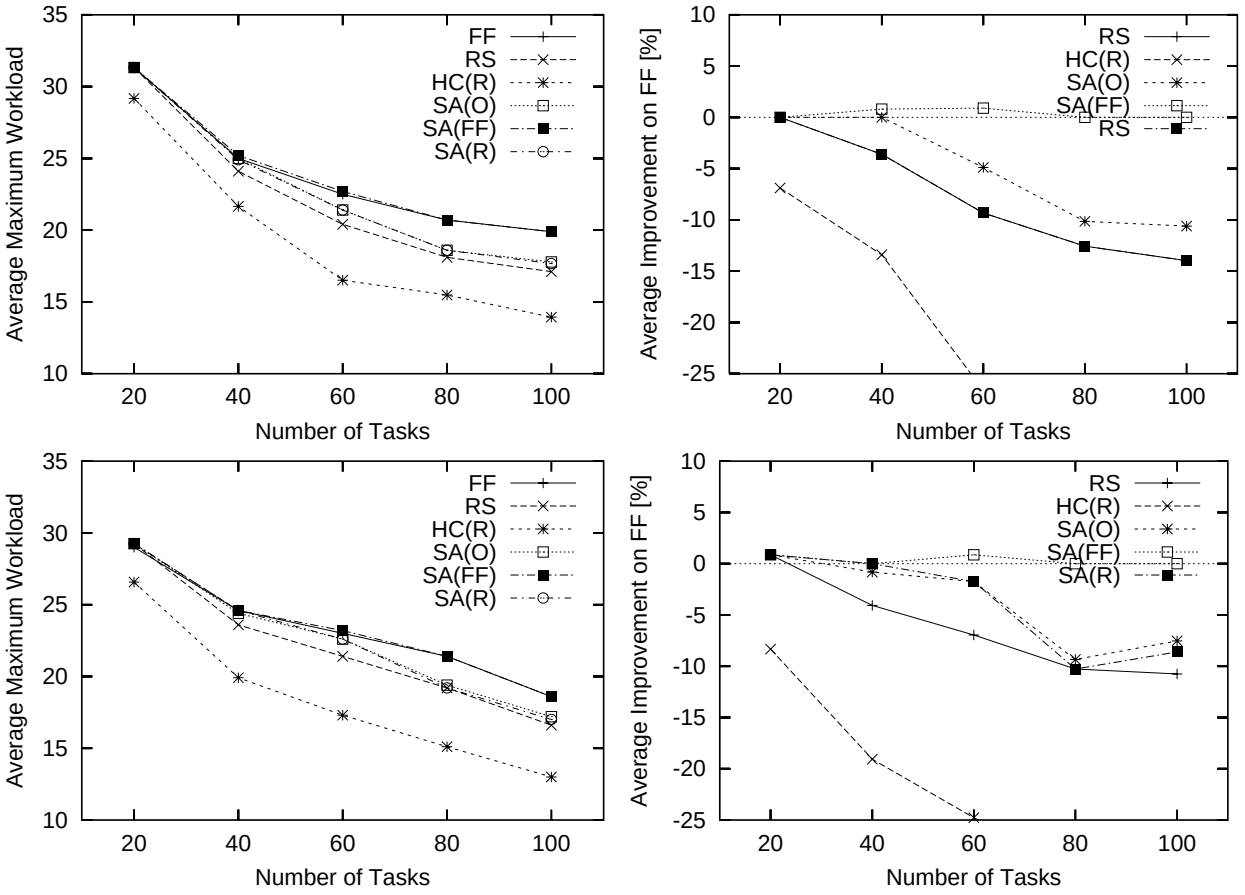

Figure 3: Performance on the second collection with input dependent and input independent tasks. The results for simulated annealing and random search are based on 100000 iterations (above) and 400000 iterations (below). The left diagrams show the average maximum workload obtained. The diagrams on the right show the average percental improvement of workload compared to first fit.

engage tasks, and 10 guide missile tasks. Without loss of generality, we assume that deadline and period for the detect task is 0.1 sec, the deadline and period for the engage task is 0.01 sec, and the deadline and period for the guide missile task is 0.05 sec. The following table compares the results for the three scenarios for simulated annealing, random search, hill climbing, and first-fit.

3 Conclusions

This paper examined the *maximum allowable workload* (MAW) problem for real-time systems with tasks having variable workloads. We have introduced several heuristic algorithms and compared their performance in extensive simulations. The results indicate that FF perform reasonably good compared to the other approaches. This suggests that FF may be a reasonable approach for resource allocation in real-time systems with variable workloads. The motivating example demonstrated the applicability of our model.

References

[1] E. H. L. Aarts, J. H. M. Korst, and P. J. M. van Laarhoven. Simulated annealing. In E. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization, 1997, John Wiley&Sons Ltd.*, pages 91–120. John Wiley&Sons, West Sussex (England), 1997.

[2] T. F. Abdelzaher and K. G. Shin. Optimal combined task and message scheduling in distributed real-time systems. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 162–171. IEEE Computer Society Press, 1995.

[3] T. F. Abdelzaher and K. G. Shin. Combined task and message scheduling in distributed real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(11):1179–1191, 1999.

[4] S. Ali, A. A. Maciejewski, H. J. Siegel, and J. K. Kim. Definition of a robustness metric for resource allocation. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003)*, 2003.

[5] L. Baccouche. Efficient static allocation of real-time tasks using genetic algorithms. In *Internal Report, Imag Intitute, Laboratoire de Genie Informatique*, 1995.

[6] M. Coli and P. Palazzari. A new method for optimisation of allocation and scheduling in real-time applications. In *Proceedings of the Seventh Euromicro Workshop on Real-Time Systems*, pages 262–269, 1995.

[7] M. DiNatale and J. A. Stankovic. Aplicability of simulated annealing methods to real-time scheduling and jitter control. In *Proceedings of the IEEE Real-Time Systems Symposium*, 1995.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Company, San Francisco, 1979.

[9] S. Gertphol, Y. Yu, S. B. Gundula, V. K. Prasanna, S. Ali, J. K. Kim, A. A. Maciejewski, and H. J. Siegel. A metric and mixed-integer-programming-based approach for resource allocation in dynamic real-time systems. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002)*, 2002.

[10] G. Greenwood, C. Lang, and S. Hurley. An evolutionary stragegy for scheduling periodic tasks in real-time systems. In *Applied Decision Technologies*, pages 171–188, 1995.

[11] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, (220):671–680, 1983.

[12] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek. On quality of service optimization with discrete qoS options. In *Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium (RTAS '99)*, pages 276–286, Washington - Brussels - Tokyo, June 1999. IEEE.

[13] C. Lee and D. P. Siewiorek. An approach for quality of service management. Technical Report CMU-CS-98-165, Computer Science Department, Carnegie Mellon University, 1998.

[14] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.

[15] M. Nicholson. Allocating and scheduling hard real-time tasks on a point-to-point distributed system. In *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems*, 1993.

[16] D-I. Oh and T. P. Baker. Utilization bounds for N-processor rate monotonic scheduling with stable processor assignment. *Real Time Systems Journal*, 15(1):183–193, 1998.

[17] D. T. Peng, K. G. Shin, and T. F. Abdelzaher. Assignment and scheduling of communicating periodic tasks in disributed real-time systems. *IEEE Transactions on Software Engineering*, 23(12), 1997.

[18] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. A QoS-based resource allocation model. In *Proceedings of the IEEE Real-Time Systems Symposium*, 1997.

[19] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. Practical solutions for QoS-based resource allocation problems. In *Proceedings of the IEEE Real-Time Systems Symposium*, 1998.

[20] F. E. Sandnes. A hybrid genetic algorithm applied to automatic parallel controller code generation. In *Proceedings of the eighth Euromicro Workshop on Real-Time Systems*, pages 70–75, 1996.

[21] Z. Tan. Producing application CPU profiles in DynBench via curve fitting, 2003. manuscript.

[22] K. Tindell, A. burns, and A. Wellings. Allocating real-time tasks (an np-hard problem made easy). *Journal of Real-Time Systems*, 4(2):145–165, 1992.

[23] Lonnie R. Welch and Behrooz A. Shirazi. A dynamic real-time benchmark for assessment of qos and resource management technology. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, pages 36–45, 1999.

[24] Y. Zhou. Execution time analysis for dynamic real-time systems. Master's thesis, School of Electrical Engineering and Computer Scienc, Ohio University, 2002.