Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)

Greedy Scheduling with Complex Objectives

Carsten Franke, Joachim Lepping, and Uwe Schwiegelshohn

***Abstract*— We present a methodology for automatically generating an online scheduling process for an arbitrary objective with the help of Evolution Strategies. The scheduling problem comprises independent parallel jobs and multiple identical machines and occurs in many real Massively Parallel Processing systems. The system owner defines the objective that may consider job waiting times and priorities of user groups. Our scheduling process is a variant of the simple and commonly used Greedy scheduling algorithm in combination with a repeated sorting of the waiting queue. This sorting uses a criterion whose parameters are evolutionary optimized. We evaluate our new scheduling process with real workload data and compare it to the best offline solutions and to the online results of the standard EASY backfill algorithm. To this end, we partition the user of the workloads into groups and select an exemplary objective that prioritizes some of those groups over others.**

I. Introduction

In this paper, we use Evolution Strategies to generate a method that specifically considers the preferences of the system owner when automatically generating an online scheduling process for Massively Parallel Processing (MPP) systems. Our scheduling problem is taken from real MPP installations: Different user submit independent, non-clairvoyant parallel jobs to the MPP system over time. The scheduling process is responsible to assign those jobs to the available identical machines of the MPP system.

Because of limited available processing time, existing online scheduling processes at real installations mainly use Greedy scheduling [1] in combination with First-Come-First-Serve and Backfilling [2] methods. Although those algorithms produce a very low utilization in the worst case [3] they work well and fast in practice [4]. Other scheduling algorithms have been subject of theoretical and simulation studies but are rarely found at real installations, see Feitelson et al. [5] and the references therein, as they often require more execution time. Therefore, our scheduling process is also based on a Greedy approach. However, the parameters are carefully chosen to consider the scheduling criterion and the workload. To this end, we use Evolution Strategies that are introduced in Section II-B.

Manuscript received on October 31, 2006. This work was financially supported by the Deutsche Forschungsgemeinschaft (DFG) as a subproject of the Collaborative Research Center 531, "Computational Intelligence", at the Dortmund University.

Carsten Franke is a former member of the Robotics Research Institute at Dortmund University and is now with SAP Research CEC Belfast, TEIC Building, University of Ulster, Newtownabbey BT37 0QB, UK (email: carsten.franke@sap.com). Joachim Lepping and Uwe Schwiegelshohn are with the Robotics Research Institute, Section Information Technology (IRF-IT), 44221 Dortmund, Germany (email: {joachim.lepping, uwe.schwiegelshohn}@udo.edu).

In real-life, system owners have different relationships to the various users or user groups of their systems. Those relationships lead to different priorities of the users and their jobs. This is particularly true as MPP systems increasingly become part of Computational Grids nowadays [6], that is, low priority users from other sites request system resources as well. Due to the existence of different user priorities that may change over time, there is a growing need for scheduling systems that can flexibly consider those priorities without reducing overall system utilization. As existing scheduling strategies are not suited to satisfactorily handle those priorities, system owners often set partitions or quotas to prevent lower priority groups from occupying too many resources. However, those restrictions tend to reduce machine utilization significantly [7].

We model an MPP system as m identical parallel machines. This model closely matches reality as differences between the nodes of an MPP system usually are not significant. Job scheduling on MPP systems is an online problem as jobs are submitted over time and the processing time p_j of job j is not available at the release date r_j. However system administrators often require users to provide estimates $\bar{p}_j$ of the processing time p_j to determine faulty jobs whose processing times exceed a rather high estimate. Therefore, the estimates are not really reliable [8]. Nevertheless, they are used for scheduling with backfilling as no other data are available [6]. Further, many parallel jobs on MPP systems are not moldable or malleable [5], that is, they need concurrent and exclusive access to $m_j \leq m$ machines during their whole execution. The user provides the value m_j at the release date r_j of the job. The completion time of job j in schedule S is denoted by $C_j(S)$. As preemption [5] is not allowed in many MPP systems, each job starts its execution at time $C_j(S) - p_j$.

Our work is based on workload traces of real installations. Such workload data include all hidden job dependencies, patterns, and feedback mechanisms. Several workloads of MPP systems are publicly available, see the standard workload archive maintained by Feitelson [9]. Unfortunately, they only partly provide user group information while owner priorities are missing completely. Therefore, this information is added to the workloads as explained in Section IV-A. As already mentioned common scheduling algorithms only support standard scheduling criteria like utilization or average waiting time [5]. To demonstrate the ability of our method to support unconventional criteria we define one in Section IV-B. Finally, we use two approaches to evaluate the results of our scheduling process:

1) Comparison with an approximation of the optimal offline result
2) Comparison with the result of a standard online schedul-

ing algorithm

Those results are presented in Section V. The paper ends with a brief conclusion.

II. Background

In this section, we first describe the general scheduling system and then explain two standard scheduling algorithms for MPP systems. Afterwards we introduce Evolution Strategies that are used for parameter optimization.

A. Common Scheduling Concepts

The scheduling system of an MPP system consists of three major components: the current schedule, the current waiting queue, and the scheduling algorithm. The current *schedule* describes the actual allocation of processor nodes to jobs. Jobs that have already been submitted to the system but did not yet start are stored within the *waiting queue*. The last component is the *scheduling algorithm* which is responsible for the assignment of jobs to available nodes.

Most scheduling algorithms use an ordered waiting queue. Often a static ordering based on submission times or runtime estimates is applied. Alternatively, the waiting queue can be reordered depending on the system state that may include the current schedule, the waiting queue, and past scheduling decisions. This reordering approach allows the consideration of dynamic information, like, for instance, the wait time of a job an thus lead to complex sorting criteria.

Scheduling algorithms mainly differ in the way they select the next job from the ordered waiting queue to insert it into the current schedule. This results in different algorithmic complexities and correspondingly different execution times. In the following, we present two standard scheduling algorithms of real MPP systems. The submission times of the jobs determine the order of the waiting queue for both algorithms. Therefore, the ordering of the waiting queue does not require any additional processing time.

1) First-Come-First-Serve (FCFS): The job on top of the queue is started as soon as enough idle machines are available to execute it. Therefore, the algorithm uses a Greedy approach. It has a constant computational complexity as the scheduler only tests whether the first job can be started immediately if a job in the schedule has completed its execution or a new job has risen to the top of the waiting queue.

2) EASY Backfilling (EASY): If there are enough idle machines to start job j on top of the waiting queue, this job is started immediately. Otherwise the algorithm uses the runtime estimates of all jobs in the current schedule to estimate the start time of j. Then it tries to find an allocation for the following jobs of the waiting queue on currently idle machines such that projected start time of j is not further delayed by using the runtime estimates of the jobs in the waiting queue. The first job in the queue that satisfies this so called backfilling condition is started immediately. Then the rest of the waiting queue is processed in the same way. EASY is restarted whenever a job of the current schedule completes its execution. Further, if EASY is not actively evaluating jobs in the waiting queue then a newly submitted job is started immediately if it satisfies the backfilling condition. EASY requires more computational time than FCFS as the scheduler considers all jobs of the waiting queue and evaluates the backfilling condition if the top job cannot be started immediately.

As EASY produces to better scheduling results than FCFS for the examined workloads it is used as the reference online algorithm within our study.

B. Evolution Strategies

During the development of the scheduling process, several parameters must be optimized regarding the previously specified priorities and the scheduling objective. This offline and nonlinear optimization problem involves recorded workloads, nonlinear sorting criteria, and a potentially nonlinear objective function. To solve it, we apply $(\mu \stackrel{+}{,} \lambda)$-Evolution Strategies [10] which are a subclass of Evolutionary Algorithms. Due to our definition of the sorting criteria in Section III-A, optimization of real valued parameters is required for which Evolution Strategies are most appropriate, see Bäck and Schwefel's comparative studies [11] on Evolution Strategies and Genetic Algorithms.

Evolution Strategies are random search methods that mimic the behavior of biological evolution. They operate on a parental population of μ individuals and apply genetic operators like selection, mutation and recombination to breed λ offspring individuals from those μ parent individuals. Based upon the evaluation results, a new population of individuals is selected. There are two variants of Evolution Strategies: In case of the (μ, λ)-Evolution Strategy, the new parents are chosen from the offspring only while the $(\mu + \lambda)$-Evolution Strategy also takes the parents into account.

Furthermore, the concept of self-adaptation enables Evolution Strategies to vary the mutation step size. Thus, in contrast to the majority of Evolutionary Algorithms, the Evolution Strategies do not require a separate tuning strategy for the optimization parameters. Therefore, Evolution Strategies can be used to explore large real valued solution spaces with few required evaluations, see Bäck and Schwefel [11]. As in our study, a single evaluation is computationally expensive the fast convergence of the optimization algorithm is of major importance.

Due to the type and the size of the optimization problem, we cannot apply standard algorithms that guarantee an optimal result. Certainly, there are other heuristics approaches that produce good results in practice, like Tabu search [12] or simulated annealing [13]. However, we do not apply these algorithms as the search space is relatively small, see Section III, and Evolution Strategies have been proven to work very well in such environments, see Beyer and Schwefel [10]. Moreover, it is not our goal to determine which algorithm is superior in solving the optimization problem but to show that online scheduling process can be improved in practice by using methods of computational intelligence. Further note that those methods are not part of the actual online scheduling algorithm

but they are only used to optimize the parameters of a standard scheduling process.

III. Methodology

For the development of an online scheduling process that supports user group prioritization defined by the system owner, we use a modified Greedy algorithm: First, jobs are inserted into the waiting queue after release as usual. Then the waiting queue is ordered using a sorting criterion with dynamic components. Finally, the job at the top of the queue is scheduled as in the FCFS algorithm, see Section II-A.1. The sorting of the waiting is repeated whenever a new job is submitted or a job in the current schedule has completed its execution. Due to this sorting process, the algorithm has a larger computational complexity than FCFS. In practice however, the processing time is rather small as the order of the jobs in the waiting queue does not significantly change from one sorting process to the next. In comparison to EASY, both the computational complexity and the processing time in practice are reduced significantly.

Greedy scheduling has already been analyzed in theoretical studies, see, for instance, Schwiegelshohn [3] and Yahyapour. In most previous studies, the waiting queue is statically ordered with the help of job weights that are input data provided by the users. However in our approach, the parameters of the sorting criterion are selected without further user specifications such that the schedule is optimized with respect to the objective of the system owner.

A. Complex Sorting Criteria

From the analysis of existing workloads as well as from experiences with parallel computers, it is known that there is a periodicity in the users' job submission behavior [14]. The diurnal cycle is an accepted and widely recognized phenomenon that appears in practically all workloads. This cycle typically shows higher activity during the day (8 am to 6 pm) and lower activity during the night (6 pm to 8 am). Normally, the activity is also lower on weekends than on work days. Furthermore, the characteristics of jobs depend on their submission time during the day. During normal work hours, more interactive small jobs are submitted while time consuming jobs are commonly released towards the end of the day as they are expected to run during the night. Therefore, we distinguish the following three classes of scheduling situations at which the waiting queue is reordered:

- weekends,
- week days between 8 am and 6 pm, and
- week days between 6 pm and 8 am.

In general the sorting criteria may consider parameters that reflect the wait time of jobs that are already submitted but not yet started, the current schedule, and past scheduling decisions. As already mentioned, MPP systems increasing participate in Computational Grids. Grid scheduling typically uses a multilayer approach [15]. For security reasons, a higher layer may not receive detailed feedback information from a lower layer including data about the current schedule. Therefore, we only take the wait time into account as a dynamic parameter as this information is always available.

Next for each of the three scheduling situations above, we determine a separate sorting criterion from a set of four criteria with several parameters:

$$f_1(j,t) = \sum_{i=1}^{5}\left[\varrho_i(j)\cdot w_i\cdot\left(K_i + a\cdot\frac{t-r_j}{\bar{p}_j} + b\cdot\frac{\bar{p}_j}{m_j}\right)\right]$$

$$f_2(j,t) = \sum_{i=1}^{5}\left[\varrho_i(j)\cdot w_i\cdot(K_i + a\cdot(t-r_j) + b\cdot\bar{p}_j\cdot m_j)\right]$$

$$f_3(j,t) = \sum_{i=1}^{5}\left[\varrho_i(j)\cdot w_i\cdot\left(K_i + a\cdot\frac{t-r_j}{\bar{p}_j\cdot m_j}\right)\right]$$

$$f_4(j,t) = \sum_{i=1}^{5}\left[\varrho_i(j)\cdot w_i\cdot\left(K_i + a\cdot(t-r_j) + b\cdot\frac{\bar{p}_j}{m_j}\right)\right]$$

With t denoting the system time, the wait time of job j is $(t-r_j)$. Further, function $\varrho_i(j)$ distinguishes between different user groups:

$$\varrho_i(j) = \begin{cases} 1, & \text{if job } j \text{ belongs to user group } i \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Exemplarily, we discuss sorting criterion $f_1(j,t)$. Each user group i has a different weight parameter w_i that is related to the priority of this user group. Parameter a represents the importance of the wait time $(t-r_j)$ of job j in relation to the estimated processing time $\bar{p}_j$ of job j. Intuitively, it is usually accepted that a job requiring more processing time may wait longer. A high value of parameter b emphasizes the estimated processing time over the degree of parallelism as it is easier to find an allocation for less parallel jobs even if jobs have the same resource consumption. While parameters a and b are independent of the user group their importance can be influenced for each user group separately with the additive parameter K_i .

Note that the selection of the sorting criteria candidates is heuristic. In this study, we used our experiences in scheduling jobs on MPP systems. The availability of specific knowledge about a particular MPP system may suggest other criteria. It is possible that the scheduling results would be different for other criteria but based on our simulations we do not expect large deviations. Moreover, a larger number of candidate functions with more parameters will lead to a significantly increased optimization effort. Finally, the selection of the sorting criteria does not limit the general applicability of our method.

Our four sorting criteria result in a total of 12 static parameters (a, b, $\{w_i, K_i$ for $i \in \{1,\ldots,5\}\}$). Further, there are three classes of scheduling states each with a separate sorting criterion. Therefore, between 33 and 36 parameters are required for a scheduler as sorting criterion $f_2(j,t)$ does not require parameter w_i. The optimal set of parameters depends on the MPP system and the workload. As already explained in Section II-B an Evolution Strategy is used to find good sets of those parameters.

IV. Exemplary Execution

To demonstrate the feasibility of our approach, we use workloads from real installations as input data and define user groups as it is not possible to extract them directly from workload data. Then we introduce an objective function that allows a comparison with existing scheduling algorithms. The parameters of our sorting criteria are trained for this objective function and the given workloads.

A. Workloads

In this study, we use six well known workload traces that were recorded at the Cornell Theory Center (CTC), the Royal Institute of Technology (KTH) in Sweden, the Los Alamos National Lab (LANL), and the San Diego Supercomputer Center (SDSC00/ SDSC95/ SDSC96). Each trace provides request and execution data of all individual job submitted during a given time period. To allow comparisons between the results derived from those traces, they are scaled to a standard machine configuration with 1024 processors by using the approach of Ernemann et al. [16], see Table I. Note that the original EASY schedule is not valid for the scaled workload. Therefore, we apply a discrete event simulation to produce the EASY schedule of the scaled workload. Although the data are rather old they include all representative patterns of job submissions which are still observed in current installations. Therefore, these workload data still suffice for our purpose. Unfortunately, user group data are at most partially available

Identifier	CTC	KTH	LANL	SDSC00	SDSC95	SDSC96
Machine	SP2	SP2	CM-5	SP2	SP2	SP2
Period	06/26/96 - 05/31/97	09/23/96 - 08/29/97	04/10/94 - 09/24/96	04/28/98 - 04/30/00	12/29/94 - 12/30/95	12/27/95 - 12/31/96
Processors (m)	1024	1024	1024	1024	1024	1024
Jobs (n)	136471	167375	201378	310745	131762	66185

TABLE I
Scaled Workload Traces from the Standard Workload Archive [9] using the scaling procedure of Ernemann et al. [16]

in those traces as current schedulers do not exploit those data. To address the is problem, we partition the individual users into different groups based on resource consumption. More specifically, we generate 5 user groups such that user group 1 represents all users with a high resource consumption whereas all users in group 5 have a very low cumulative resource demand. To this end, we determine the ratio between the total resource consumption of user u

$$\mathrm{RC}_u = \sum_{j \in \tau} p_j \cdot m_j \cdot \varpi_u(j) \tag{2}$$

and the total resource consumption of all users

$$\mathrm{RC} = \sum_{j \in \tau} p_j \cdot m_j. \tag{3}$$

Function $\varpi_u(j)$ is similar to function $\varrho_i(j)$ and specifies the relationship between a single job j and user u:

$$\varpi_u(j) = \begin{cases} 1, & \text{if job } j \text{ belongs to user } u \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

The relation between the above defined ratio and the user groups is described in Table II. Our selection seems to be

User Group	1	2	3	4	5
$\mathrm{RC}_u/\mathrm{RC}$	$> 8\ \%$	$2 - 8\ \%$	$1 - 2\ \%$	$0.1 - 1\ \%$	$< 0.1\ \%$

TABLE II
The Assignment of Users to User Groups Depending on the Corresponding Resource Consumption Ratio

appropriate as machine utilization and job submission pattern are often similar among users of the same user group in real installations. But any other selection of user groups is possible as well and does not affect the applicability of our approach.

B. Objective Functions

Due to the lack of a scheduling process that supports priority functions, appropriate scheduling objectives are not used in real installations. Therefore, we cannot extract such an objective from the given workloads and must define one. The system owner is generally free to select any objective function. This choice may only influence the processing time of parameter training. Nevertheless, there are a few criteria commonly used in scheduling policies [5]. The utilization (UTIL) of an MPP system during the execution of job set τ reflects the system owner's point of view and covers the time period beginning with the start time of the first job in this set ($\min_{j \in \tau}\{C_j(S) - p_j\}$) and ending with the completion time of the last job in this set ($\max_{j \in \tau} C_j(S)$). Formally, it is the ratio of machines busy with jobs of this set to available machines during this time period:

$$\mathrm{UTIL} = \frac{\sum_{j \in \tau} p_j \cdot m_j}{m \cdot \left(\max_{j \in \tau} \{C_j(S)\} - \min_{j \in \tau}\{C_j(S) - p_j\} \right)} \tag{5}$$

The Average Weighted Response Time (AWRT) considers the completion time of all jobs in a set and allows different prioritization of the individual jobs. If, we weight all jobs with their resource consumption $(p_j \cdot m_j)$ as suggested by Schwiegelshohn and Yahyapour [3] then no jobs has a higher priority than any other:

$$\mathrm{AWRT} = \frac{\sum_{j \in \tau} p_j \cdot m_j \cdot (C_j(S) - r_j)}{\sum_{j \in \tau} p_j \cdot m_j} \tag{6}$$

Intuitively, a short AWRT states that on average the users do not wait long for their jobs to complete in relation to the resource consumption of the jobs. There is a strong correlation between AWRT und UTIL.

However, AWRT can also be defined for each user group separately. This results in the group specific objectives AWRT_i with $i \in \{1, 2, \ldots 5\}$:

$$\mathrm{AWRT}_i = \frac{\sum_{j \in \tau} p_j \cdot m_j \cdot (C_j(S) - r_j) \cdot \varrho_i(j)}{\sum_{j \in \tau} p_j \cdot m_j \cdot \varrho_i(j)} \quad (7)$$

As there is no real life scheduling objective available from the workload traces we must again define such an objective function f_{obj}. To this end, we use those basic objective functions introduced above. With the help of an appropriate selection, we can easily improve our scheduling results in comparison with a standard online scheduling algorithm. For instance, the backfilling of EASY favours sequential jobs which are particularly common in user groups with high resource consumption. Thus giving priority to user groups with small resource demands will result in a bad performance of EASY and give our new algorithm the chance to excel. However, it is not likely that a system owner will particularly favour those user groups that contribute the least to a high utilization of his machines. Therefore, we select an objective function that gives higher priority to those user groups with a higher resource consumption. This is the implicit objective of EASY and allows a fair comparison. Moreover in this study, we restrict ourselves to two basic objective functions as this allows a better graphical representation of the results within an approximated Pareto front, see Section V-C. Those thoughts lead to the objective function

$$f_{obj} = 10 \cdot \mathrm{AWRT}_1 + 4 \cdot \mathrm{AWRT}_2. \quad (8)$$

The choice of another objective function may lead to different results but does not affect the feasibility of our methodology.

C. Training of Parameters

During training, the objective function f_{obj} determines the scheduling quality. Our procedure uses all combinations of the four sorting criteria with the three scheduling situations. Hence, we optimize twelve different systems and finally for each situation, we select the system that performs best.

For the determination of the required 36 object parameters, we use a (μ+λ)-Evolution Strategy with a real-value coding as explained in Section II-B. An individual is characterized by the set of object parameters and a set of strategy parameters. The fitness of this individual is the value of the complex scheduling objective f_{obj} obtained from the schedule that is generated from the training workload with the object parameter set of the individual. Each single optimization involves 100 generations. We set $\mu = 15$ and $\lambda = 105$ matching Schwefel's recommendation of $\mu/\lambda = 1/7$ [11]. In order to reduce the search space, the values of the 36 object parameters are limited to $w_i \in [0, 1]$, $a \in [0, 1]$, $b \in [0, 1]$, and $K_i \in [0, 5]$. As suggested by Beyer and Schwefel [10] we further apply a self-adaptation of the mutation step size. This self-adaptation is controlled by strategy parameters of the mutation distribution, which are internally evolved with an Evolution Strategy rather than being predetermined by some exogenous scheme. As there is a separate mutation strength for each object parameter we also need a set of 36 strategy parameters. Furthermore, within our Evolution Strategy we use *discrete* recombination with all μ parents for the object parameters whereas *intermediate* recombination with 2 randomly selected parents is applied to the strategy parameters, see also Beyer and Schwefel [10].

The afore mentioned schedule evaluation with a discrete event simulation is extremely time consuming (about 4 hours on a high end PC) as most workload traces consist of more than 100,000 jobs which have been submitted over a time period of approximately one year, see Table I. Therefore, fast convergence of the Evolution Strategy is of great importance. Hence, we parallelized our evaluations and computed them on a 120 processor cluster. This is possible because all simulations are completely independent from each other. Then the optimization of the parameter set for a single workload takes approximately two weeks. Nevertheless, more than three months of calculation were required to obtain all the results presented in this paper.

Note that the given effort estimates are only related to the parameter training of our Greedy algorithm. This training is only required if the composition of the workload changes. As already mentioned the execution of our scheduling algorithm in the runtime environment does not use Evolution Strategies and is faster than the execution of EASY. Therefore, our scheduling strategy can compete well with approaches used in existing MPP systems.

D. Generation of Algorithms

In this study, we use two different approaches to automatically generate a Greedy strategy for a given scheduling objective.

- **Single Workload Training**: We select the CTC workload trace and train the parameters only for this workload. Note that any of the available workloads can be chosen but the required computational effort does not allow a parallel generation of algorithms for all workloads. However, small scale experiments for other workloads give indications of very similar results. Then we apply the Greedy algorithm with the trained parameters on the same workload. The result determines the improvement potential of this approach. Note that our process is subject to several restrictions:
  1) We use a Greedy approach without backfilling to keep the processing time of the algorithm low.
  2) No feedback from the previously scheduled jobs and the current schedule is considered to support multi layer Grid scheduling.
  3) The choice of the basic sorting criteria is not optimized.

  Therefore, it is not likely that the scheduling result of our algorithm will be very close to the optimal offline result. But we expect an improvement in comparison to EASY. Then we apply the Greedy algorithm with this parameter set also to the other workload traces and individually

compare the resulting online schedules to the schedules produced by the application of EASY using our scheduling objective. As the parameter set is only trained on the CTC workload we expect only an improvement of the results for those workloads who have a similar job composition as the CTC workload.

- **Multiple Workload Training**: We generate a parameter set that minimizes the sum of the objective values of all workload traces. Again for each workload, we compare the resulting online schedule to the corresponding EASY schedule. With this approach, we want to determine how well a single static parameter set can work for all workloads despite their differences in job composition.

V. Evaluation

In addition to the direct comparison of two online schedules, we also determine the quality of the produced online schedules in relation to the schedules of an approximate Pareto front of feasible schedules that was generated offline from the workload traces [17]. This Pareto front approximation is executed for each workload trace separately and is based on the seven basic objective functions: UTIL, AWRT, and $AWRT_1$ to $AWRT_5$. Similar approaches were presented by Balicki and Kitowski [18] but for less dimensions. Although the generation of an approximate Pareto front is not subject of this paper, two restrictions must be noted:

1) For real workloads, we are only able to approximate Pareto fronts. Therefore, schedules of this front are not guaranteed to be real Pareto optima.
2) The schedules are generated offline. Online methods may not be able to achieve as good results due to online constraints.

A. Single Workload Training

First, we address the results when the parameters are trained by using only the CTC workload trace. Table III shows the schedule results of our Greedy algorithm and EASY for the objective functions UTIL and $AWRT_1$ to $AWRT_5$. Note that the utilization is not affected by giving a higher priority to user groups 1 and 2 although our objective function does not include UTIL. The relations between the $AWRT_i$ values of both schedules are also given in Figure 1. Obviously, our Greedy algorithm achieves the desired prioritization as $AWRT_1$ and $AWRT_2$ improve with respect to EASY. The value of objective function decreases by more than 9 %, see Figure 2. However, the values of $AWRT_3$ to $AWRT_5$ of the Greedy schedule are much larger compared to the EASY schedule as those user groups are not considered in the objective function. Clearly, the low-priority users must pay the prize as overall utilization remains unchanged.

The generated Greedy strategy uses the sorting criterion $f_2(t, j)$. This sorting criterion performes best in all of our different optimization approaches with the Greedy algorithm. Therefore, $f_2(t, j)$ can serve as a starting point for a study on appropriate sorting criteria.

Method	$AWRT_1$	$AWRT_2$	$AWRT_3$	$AWRT_4$	$AWRT_5$	UTIL
GREEDY	52755.80	61947.65	56275.18	54017.23	35085.84	66.99
EASY	59681.28	64976.07	50317.47	46120.02	31855.68	66.99

TABLE III
Absolute results of AWRT (in Seconds) and UTIL (in %) for the CTC workload trace and single workload training

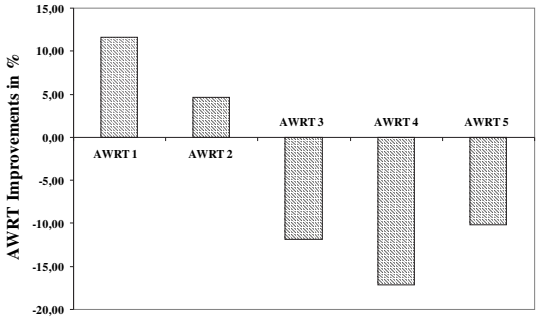

Fig. 1. AWRT improvements of the Greedy algorithm versus EASY for the CTC workload and single workload training

The Greedy algorithm with the CTC trained parameter set is then applied to the other workload traces. Figure 2 shows

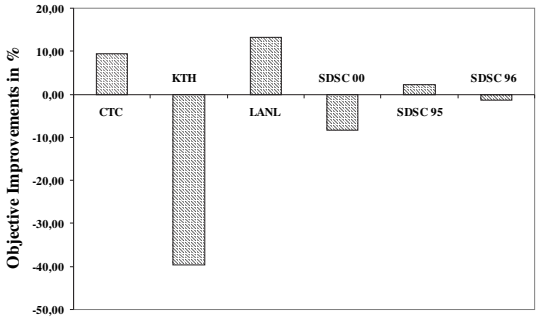

Fig. 2. Objective improvements of the CTC trained Greedy algorithm applied to other workloads

that the algorithm performs well for the CTC, LANL, and SDSC95 workload traces but significantly fails for the KTH and SDSC00 workloads. For the SDSC96 workload trace it produces slightly worse results than EASY. This indicates that the job composition of the workloads CTC, LANL, and SDSC95 is rather similar, while there are significant differences to the composition of the other workloads. Note that this is an evaluation of the workload directly based on scheduling algorithms while other workload characterizations use statistical methods [14] and hope that similar statistic properties will result in a similar behavior during scheduling.

B. Multiple Workload Training

Compared to Figure 2, Figure 3 shows that training on multiple workloads results in a significant but smaller improve-

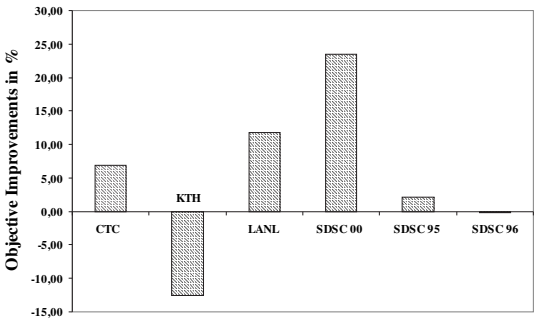

Fig. 3. Objective improvements of the multiple workload trained Greedy algorithm applied to all workloads

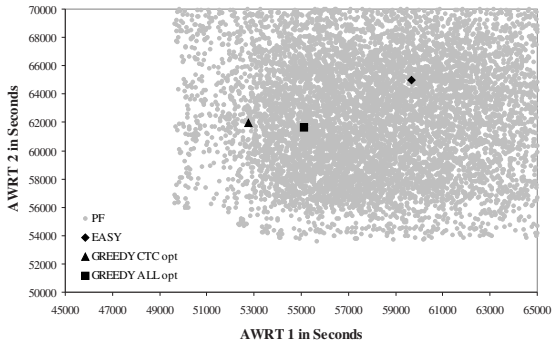

Fig. 4. Positions of the schedules produced by EASY, CTC trained Greedy, and multiple workloads trained Greedy algorithms in the ($AWRT_1$,$AWRT_2$)-chart in comparison to the 7-dimensional Pareto front approximation of the CTC workload

ment for the CTC workload trace while the performance is better for all other workload traces. The influence of the other workloads is particularly obvious for the SDSC00 workload trace: When the parameter set is trained only with the CTC workload trace the EASY schedule is clearly better than the Greedy schedule. However, when all workload traces are used during training the SDSC00 Greedy schedule shows the largest improvement over the corresponding EASY schedule. But the results also show that there is no single parameter set that produces good scheduling results for a given objective and all workloads. This suggests the use of more sophisticated hyperheuristic approaches, see for example Burke et al. [19], [20], if the workload composition is changing. Those approaches may select between parameter sets trained on different workloads. Further, they need an additional feedback mechanism to identify changes in the workload composition.

C. Results Compared to Pareto Front Approximation

In a graphical representation of the Pareto front approximation, each point corresponds to a feasible schedule that is not dominated by any other generated feasible schedule of the used workload trace. Note that those points cover an area in our two-dimensional chart as our chart only shows a projection of the actual seven-dimensional Pareto front approximation. As the Pareto front was generated offline and our online schedule is subject to various restrictions, see Section IV-D, it is unlikely that schedules of this front can actually be produced by our online scheduling systems. Therefore, we refer to this front as an lower bound of the best achievable solution. Although we do not know the real Pareto front, the high density of our approximation indicates a high quality of the approximation, see Figures 4 and 5. Those figures show a chart using the AWRT objectives that are part of the scheduling objective. Therefore, the distance to the appropriate border position in the lower/left corner of the area covered by the Pareto front approximation represents an upper bound of the possible improvement. This must be seen in relation to the distance between the positions of the Greedy schedule and the EASY schedule in the chart. As EASY does not explicitly favour any user group, the corresponding AWRT values are likely to be located farther away from the mentioned border position. In each figure, we highlight the schedules obtained by the single workload (CTC) trained Greedy algorithm, the multiple workload trained Greedy algorithm, and EASY.

Figure 4 represents the CTC workload and visually shows that both Greedy algorithms produce significant improvements, see also Figures 1 and 2. Due to the scheduling objective, the improvement is larger for $AWRT_1$ than for $AWRT_2$. While there may still be some room for further improvement, the area to the left of the CTC trained Greedy schedule exhibits significantly less density. Therefore, it is not really clear whether those schedules can actually be reached with an online Greedy algorithm. Figure 5 based on the

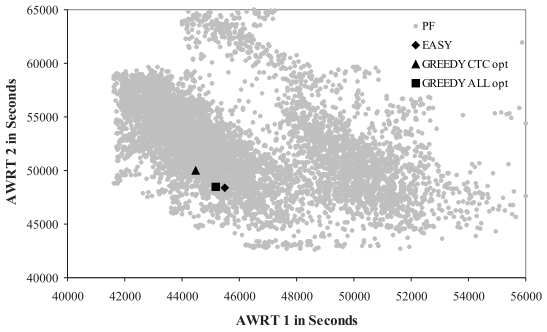

Fig. 5. Positions of the schedules produced by EASY, CTC trained Greedy, and multiple workloads trained Greedy algorithms in the ($AWRT_1$,$AWRT_2$)-chart in comparison to the 7-dimensional Pareto front approximation of the SDSC 95 workload

SDSC95 workload trace shows that both Greedy algorithms produce similar results that are not clearly better than the EASY schedule, see also Figures 1 and 2. However, the area between the Greedy schedule and the lower/left border line of the Pareto front projection is not as dense as in the CTC case. This may indicate that improvements over EASY are only possible if the parameter set is specifically trained with

this workload.

VI. Conclusion

We have proposed a methodology to generate online Greedy scheduling algorithms for MPP systems that are computational inexpensive and support owner defined scheduling objectives. The objective may express preferences of user groups that typically exist at real MPP installations. To our knowledge, those scheduling algorithms do not yet exist. The Greedy algorithm needs a parameter set that determines sorting criteria for the waiting queue of the scheduling process. Those parameter sets are evolutionary optimized in an offline process using recorded workload traces.

We have applied our approach to real workload traces that were scaled to quantitatively compare the obtained schedules. We were able to obtain significant improvements when the trained parameter set was used on the same workload. However, it became also apparent that the composition of the workload has a significant influence on the performance of the algorithm. The algorithm produced excellent results on some additional workloads while it clearly failed on others. This may provide us with a different approach of workload characterization in the future.

It was the goal of this study to show that highly flexible scheduling processes are possible that require the same processing power as standard algorithms and produce better results. However, we also like to state that better schedules can be produced if more processing power is available and some of the online constraints are relaxed, see, for instance, Franke et al. [21].

References

[1] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Annals of Discrete Mathematics*, vol. 15, pp. 287–326, 1979.

[2] D. A. Lifka, "The ANL/IBM SP scheduling system," in *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP95)*, ser. Lecture Notes in Computer Science (LNCS), vol. 949. Springer, 1995, pp. 295–303.

[3] U. Schwiegelshohn and R. Yahyapour, "Fairness in parallel job scheduling," *Journal of Scheduling*, vol. 3, no. 5, pp. 297–320, 2000.

[4] D. G. Feitelson and A. M. Weil, "Utilization and predictability in scheduling the IBM SP2 with backfilling," in *Proceedings of the 12th International Parallel Processing Symposium and the 9th Symposium on Parallel and Distributed Processing*. IEEE Computer Society Press, 1998, pp. 542–547.

[5] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP97)*, ser. Lecture Notes in Computer Science (LNCS), vol. 1291. Springer, 1997, pp. 1–34.

[6] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Parallel job scheduling — a status report," in *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP04)*, ser. Lecture Notes in Computer Science (LNCS), vol. 3277. Springer, 2004, pp. 1–16.

[7] D. G. Feitelson and M. A. Jette, "Improved Utilization and Responsiveness with Gang Scheduling," in *Proceedings of the 3rd Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science (LNCS), D. G. Feitelson and L. Rudolph, Eds., vol. 1291. Springer, 1997, pp. 238–261.

[8] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "Are user runtime estimates inherently inaccurate?" in *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP05)*, ser. Lecture Notes in Computer Science (LNCS), vol. 3277. Springer, April 2005, pp. 253–263.

[9] D. G. Feitelson, "Parallel workload archive," http://www.cs.huji.ac.il/labs/parallel/workload/, October 2006.

[10] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies – A comprehensive introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, 2002.

[11] T. Bäck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 1–23, 1993.

[12] A. Hertz, E. Taillard, and D. de Werra, *Local Search in Combinatorial Optimization*. Wiley, 1997, ch. Tabu search, pp. 121–136.

[13] E. H. L. Aarts, J. H. M. Korst, and P. J. M. van Laarhoven, *Local Search in Combinatorial Optimization*. Wiley, 1997, ch. Simulated annealing, pp. 91–120.

[14] A. B. Downey and D. G. Feitelson, "The elusive goal of workload characterization," *Performance Evaluation Review*, vol. 26, no. 4, pp. 14–29, 1999.

[15] U. Schwiegelshohn and R. Yahyapour, *Grid Resource Management - State of the Art and Future Trends*. Kluwer Academic Publishers, 2003, ch. Attributes for Communication Between Grid Scheduling Instances, pp. 41–52.

[16] C. Ernemann, B. Song, and R. Yahyapour, "Scaling of workload traces," in *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP03)*, ser. Lecture Notes in Computer Science (LNCS), D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds., vol. 2862. Springer, October 2003, pp. 166–183.

[17] C. Ernemann, U. Schwiegelshohn, M. Emmerich, L. Schönemann, and N. Beume, "Scheduling algorithm development based on complex owner defined objectives," SFB 531, CI-Report, University of Dortmund, Tech. Rep. 191/05, January 2005.

[18] J. Balicki and Z. Kitowski, "Multicriteria evolutionary algorithm with tabu search for task assignment," in *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization (EMO01)*, ser. Lecture Notes in Computer Science (LNCS), E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, Eds., vol. 1993. Springer, 2001, pp. 373–384.

[19] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu, "Graph-based hyper-heuristic for timetabling problems," *European Journal of Operational Research*, vol. 176, no. 1, pp. 177–192, 2007.

[20] E. K. Burke, S. Petrovic, and R. Qu, "Case based heuristic selection for timetabling problem," *Journal of Scheduling*, vol. 9, no. 2, pp. 115–132, 2006.

[21] C. Franke, J. Lepping, and U. Schwiegelshohn, "On advantages of scheduling using genetic fuzzy systems," in *Proc. 12th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, ser. Lecture Notes in Computer Science (LNCS), E. Frachtenberg and U.Schwiegelshohn, Eds., vol. 4376. Springer, 2006, pp. 68–93.