How to Design a Job Scheduling Algorithm

Uwe Schwiegelshohn

Robotics Research Institute, TU Dortmund University, 44221 Dortmund, Germany
http://www.irf.tu-dortmund.de

Abstract. We discuss the components of a general approach to design algorithms for resource management in parallel processing systems. Starting from the observation that in this area the average influence of research publications on practical implementations is negligible, we address the three main categories constraints, objectives, and evaluation. For each category, we describe common approaches and restrictions and give some general rules that should be followed when presenting a new algorithm for resource management. As an example we present a resource management method for the IaaS model of Cloud Computing that extends the spot instance approach of Amazon. For this example, we first discuss technical, organizational, and usage constraints based on existing concepts and research results. Then we briefly describe resource management objectives from the viewpoint of a provider. After presenting our algorithmic concept, we show that an evaluation with theoretical means can also yield meaningful results in practice.

1 Introduction

The increasing demand to handle large amounts of digital data in many scientific and commercial areas has led to growing importance of data centers as these centers are supposed to provide better service at lower costs. But as Kaplan, Forrest, and Nadler [8] have already pointed out in 2008, "data center inefficiency is a widespread and growing concern". Therefore, it is not surprising that many conferences and workshops on high performance or high throughput computing devote sessions to the topic *resource management and scheduling*. But although there is a plethora of papers suggesting various new algorithms for job scheduling on parallel processor systems only very few of them are used in real machines. Many systems still apply methods that have been developed in the 90s, like EASY backfilling, see Lifka [11]. There can be only two reasons for this discrepancy: either there is a lack of communication between researchers and system engineers preventing the use of suitable algorithms in practice or the proposed algorithms are not applicable in real systems. Based on our experience in organizing workshops addressing this area for a long time, we have got the impression that practitioners follow workshops and conferences and are aware of the research presented there. Therefore, the publications must do a better job convincing these practitioners about the benefits of the approaches and research work must better consider the constraints and objectives that are relevant in practice.

With this paper, we want address these challenges by presenting some rules on constraints, objectives, and evaluation. We suggest that future papers on resource management for computer systems consider them in order to point out the applicability and the benefit of new scheduling algorithms. Some of these rules may look rather simple but a review of many scheduling papers shows that they are often ignored. The rules are supposed to be used for research work that emphasizes relevance in practice while they do not necessarily apply to basic research papers that only mention some vague reference to a potential future applicability of the developed algorithms. As an example we show the application of these rules when developing a method to manage computing resources used in the Infrastructure-as-a-Service (IaaS) model of Cloud Computing. This method is an alternative to Amazon's use of spot instances.

The further outline of the paper is as follows. First, we distinguish three different types of constraints that occur in practical job scheduling problems and show how we address them in our IaaS example, see Section 2. In Section 3, we focus on the objectives of job scheduling problems and briefly discuss the use of common theory objectives and the handling of multi-objective problems with the help of our IaaS example. Then we introduce our new allocation algorithm that can be considered as an alternative to Amazon's spot instances. In Section 5, we analyze the advantages and disadvantages of the three main evaluation approaches. We pick the theoretical approach for our example to show that theoretical methods can be beneficial for practical problems.

2 Constraints

The operation of a data center is subject to several technical, organizational, and usage constraints. These constraints define the solution space of a problem, that is, a solution is only valid if all constraints are satisfied. As in other areas of science, it is not always necessary to consider all constraints when developing a new algorithm. Clearly, we can ignore constraints if their omission does not change the solution space. Often we can also apply a simple model with different and simpler constraints if this model leads to a small reduction of the solution space. Such decisions must be clearly communicated in a paper. In particular, we must show or at least state the influence of an omitted constraint on the solution space. To determine the influence of a constraint, it may be beneficial to address evaluation aspects already in an early stage. Also there may be some commonly encountered constraints that can be relaxed and removed due to new technical results or due to a proposed different organizational structure. Such modification extends the solution space and may allow a better result.

In the remaining parts of this section, we discuss constraints in our IaaS example. IaaS is a basic service model of Cloud Computing and constitutes a market consisting of providers and customers. An IaaS provider owns IT-infrastructure and offers it to his customers. An IaaS customer wants to lease a computer infrastructure to avoid the effort and the expenses of handling his own computer infrastructure. In general, the IaaS customer expects some service

guarantee when paying for the infrastructure. IaaS customers may run their own applications on the leased hardware or there may be a third partner who uses the software of IaaS customers and the hardware of IaaS provider together with his own input data.

2.1 Technical Constraints

In our example, we consider technical constraints regarding availability of application information, sharing of the physical infrastructure, and power management of the system.

Many scheduling algorithms for computer systems are clairvoyant, that is, they require execution details of the application. Often the precise execution time of an application is not known a priori. Then users may be required to provide the estimated execution time of their applications. Some scheduling algorithms consider these estimates to be exact although it has been shown that deviations are often rather large, see Lee et al. [10]. Therefore, this information is of limited benefit for resource management. In an IaaS scenario, there is little chance to obtain reliable information since sometimes input data are not under control of the IaaS customer.

IaaS providers presently favor space sharing over time sharing. Pure space sharing may lead to potential inefficiencies as long running applications with low priority must be terminated to free resources for new high priority customer requests unless high priority jobs are rejected or there is a sufficient amount of overprovisioning. Time sharing in a computer infrastructure is technically achieved by context switching between different virtual machines, that is, the execution of a virtual machine on a physical resource is preempted and later resumed on a possibly different physical resource. To monitor compliance with a service guarantee, appropriate tools must be installed by the IaaS provider. Since these tools use the same physical resource as the application and require the availability of the resource repeatedly for a brief period of time, time sharing must be applied for these tool. Therefore, tools for system management consume a share of the physical resources, the so called *system overhead*. This overhead must be taken into account by any resource management algorithm. Although the resource consumption of these tools is not constant it can be estimated rather reliably. Therefore, we assume in this example that the provider knows at least the distribution of the amount of computing power that is required for system management tasks like service monitoring, context switching, and book keeping. Then he can select the maximum amount of system overhead that a resource management system must be able to tolerate in a time interval. We consider this value to be a constant that reduces the available resources. Therefore, we ignore the system overhead and simply assume a *slower* physical resource without system overhead. In practice, an overestimation of the system overhead has not negative impact on resource management as it is not explicitly incorporated in technical components. It will only produce allocation completion times that are earlier than the corresponding deadlines. But an unnecessary large

margin for system overhead may reduce interest of potential customers and put the provider in a disadvantageous position in comparison to his competitors.

Due to present processor architectures, we must distinguish between context switching without core migration, context switching with migration to another core on the same processor, and context switching with migration to another core on a different processor since these alternatives have different context switching penalties. Strong et al. [16] point out methods to support fast context switching such that context switching with migration is not only applicable to handle processor failure but also to improve resource management. Similarly, Mars et al. [13] have suggested the use of the so called *Bubble-Up* methodology to reduce context switching penalties and propose improving resource management by allowing colocation of interactive and batch resources contrary to the current policy in many large data centers. Therefore, we consider time sharing for some applications.

Energy expenses are a significant part of the total expenses in a data center. Idle but active resources consume power without yielding a direct benefit to the provider. But the IaaS provider must accept some overprovisioning of resources to meet the agreed quality of service in case of a machine failure. Since the total resource demand of all customers is not constant particularly if most customers are located in the same time zone, an IaaS provider may also accept additional overprovisioning to handle peak demand without rejecting customer requests. In a situation of low customer demand the provider can use two approaches to reduce his energy expenses:

- Dynamic voltage and frequency scaling (DVFS)
- Shutting down of idle resources in combination with application migration

DVFS has not received much interest by IaaS providers as it strongly depends on the individual application. A provider initiated change of the processing speed may also lead to a violation of the service agreement. Moreover, simulations have shown that local changes of power consumption may lead to thermal problems in the system, see, for instance, Ibrahim et al. [7]. Therefore, we focus on migrating applications such that some racks can be powered down while the load is balanced for the active racks.

2.2 Usage Constraints

In an IaaS market the provider must consider the demand of his customers. To this end, it is necessary to analyze the applications that are typically running on an IaaS system. Although most Cloud providers do not publish utilization data of their clusters, there are a few publications that provide some data and analysis methodologies, like, for instance, some statistics on Cloud jobs based on Google's clusters, see, for instance, Mishra et al. [14]. For the IaaS scenario we use a simplification of the responsiveness classification of web jobs by Cirne and Frachtenberg [2] in our example. We will later show that this simplification is sufficient for our purpose. In this paper, we distinguish three types of applications that can be combined with any resource type of the IaaS provider:

Interactive This application requires the system to be available for processing within a very brief period of time. For instance, web front ends belong to this type of application. In order to maintain a sufficiently high responsiveness for resources with interactive applications, time sharing with migration of such application is usually avoided and only performed in case of machine failure.

User-facing This application does not require an immediate user interaction but the user expects the result as soon as possible. Animoto's[1] rendering of images is an example for such application.

Batch This application needs a significant amount of computing over an extended period of time but has no tight deadlines. Re-indexing a database or other management jobs are examples for this type of application. Such applications may be initiated by a customer or by the provider himself and typically have a low priority.

In the following, we characterize a resource based on the application it is executing, that is, we speak of interactive, user-facing, and batch resources. For interactive resources, we can distinguish two types:

- basic interactive resources must always be running to answer sporadic requests,
- flexible interactive resources are leased and released on demand but may incur some set-up penalty.

Typically, basic interactive resources are leased over an extended period of time representing a simple form of outsourcing, that is, the customer shifts the task of IT management to the provider and benefits from economy of scale effects. Depending on the demand a basic interactive resource may be idle for some time. Although such idleness may be a tempting target for resource management it must be considered that the customer has leased the resource. The customer is free to use an idle basic interactive resource for other applications but if the IaaS provider wants to exploit the resource for another purpose, like improving energy efficiency, the consent of the customer and the observance of the service guarantee are required. Therefore, a basic interactive resource produces a static constraint from the viewpoint of an IaaS resource management system in this paper. Similar to flexible interactive resources, customers are interested in leasing user-facing and batch resources on-demand to handle varying workloads. This way the customer avoids overprovisioning of IT resources while the IaaS provider may achieve an acceptable load balance by exploiting the different workload demand patterns of his customers.

2.3 Organizational Constraints

In the IaaS market economy, provider and customer conclude a service contract based on an offer of the provider. Due to the large number of customers, a

[1] animoto.com

typical IaaS customer has little power to negotiate his contract but must accept one of the standard offers similar to a mobile phone customer. To be successful in the market and to attract many customers, IaaS providers try to satisfy the different demands of the customers when defining their offers. For a our problem, we discuss whether it is possible to define IaaS offers such that they support an efficient resource management in addition to serving the purpose of attracting and satisfying customers. In this approach, the offer is not a fixed constraint but an element of the solution space that is subject to several organizational constraints.

An IaaS offer specifies the price that the customer must pay for the provided resources and a guaranteed quality of service including penalties for violating these guarantees. It is based on a small number of different tpyes of virtual instances that usually include processor specification, amount of memory, amount of storage, and network performance[2]. In these instances, there is a coarse granularity regarding each resource type and often a close relationship between the different resource types: a large number of cores is combined with more memory and better network performance. In a large data center the total number of resources for each instance type is very large and most data centers provide resources to their main customers using space sharing. Therefore, we assume in our study only a single type of instance with a single type of resource to which we refer as machine. This approach corresponds to a fixed partitioning of the resources of the data center. Similar to a car rental company, it may be possible to additionally improve efficiency by allocating a more powerful instance to a customer request without additional costs if the requested instance is not available.

In the IaaS scenario, there are typically two aspects of a service guarantee:

Availability The customer is guaranteed that the system is available at least for a specified percentage during a specified time frame, for instance, 99% availability each month.

Responsiveness The customer is guaranteed to receive a certain amount of physical computer resources within a given (brief) time interval.

Guarantee values are set by the IaaS provider as part of his offers. The availability condition is a customer protection against long term system failure. It indirectly affects resource management as additional redundant machines must be available to compensate possible machine failure, see Cirne and Frachtenberg [2]. Usually, the determination of an appropriate amount of overprovisioning can be separated from other resource management tasks. Therefore, we consider the amount of overprovisioning to be fixed in the IaaS example and do not address the availability constraint separately. Since these redundant machines are idle most of the time, a resource management system may at least partially use them for batch applications with time sharing.

The responsiveness guarantee for a virtual machine can formally be expressed by the ratio between a time interval Δt and the total time that a physical resource

[2] as an example see aws.amazon.com/ec2/instance-types/

has been allocated to a virtual machine during Δt. This ratio is not allowed to exceed the so called *stretch* or *slack* factor f defined in the contract. The system overhead discussed in Section 2.1 is a reason for a stretch factor that is greater than one. A lower stretch factor may be more attractive for a customer and may allow a higher price. The impact of the stretch factor on resource management will be discussed later. For long term leases, responsiveness can be combined with availability by defining that a machine is not available in a time interval if the response time guarantee is not always met during this interval. User-facing and batch resources primarily differ with respect to their responsiveness, that is the length of the time interval and the amount of resources allocated within the time interval.

Most IaaS providers offer physical resources in an on-demand fashion for user-facing and batch resources and use two approaches to improve efficiency: usually there is a minimum amount or even a quantum of resources that must be leased[3] and the provider will deliver a certain amount of resources during a given time interval at his discretion for user-facing and batch applications, that is, the provider decides when to deliver this guaranteed amount of resources during the time interval. Without these restrictions, on-demand offerings are not attractive: either the provider keeps many resources in stand-by for a worst case situation of high demand or the customer runs a high risk of a request being rejected due to unavailability of resources. Therefore, the restrictions represent a trade-off for the customer and the provider. The customer accepts a slightly increased responsiveness and possibly a small amount of overprovisioning while the provider benefits from the different use patterns of his customers. In general, the provider accepts some overprovisioning in order to avoid alienating his customers by rejecting their requests since there is a strong lock-in effect in Cloud Computing and lost customers may be gone forever.

Although no running time information is available the quantum lease approach produces additional information for resource management. But it must also be considered that resource allocations can be extended on request of the customer. Altogether, we have a clairvoyant online problem.

In general the IaaS provider is interested in saving energy but he also does not want to frequently power up and down resources to avoid the associated overhead and to comply with the energy contract since electrical utility providers usually require data center like their other large customers to estimate their power consumption in advance and to pay penalties in case they do not match the estimate. Due to their long lease, it is relatively easy to allocate basic interactive resources. The same is true for batch resources due to their flexibility based on their long responsiveness. A sudden increase in the demand of flexible interactive or user-facing resources may be managed by postponing some batch resources. If this approach is not sufficient new physical resources must be powered up and a penalty for a wrong energy estimation may occur. A sudden decrease in the demand may be compensated by allocating batch resources earlier than originally planned. The explained benefit of the usefulness of batch resources for

[3] Amazon uses an instance hour for its Elastic Computing Cloud (EC2).

resource management is also demonstrated by Amazon's introduction of its spot instances. Altogether, there are opportunities for a resource management system to improve efficiency while avoiding frequently switching a resource from active mode to inactive mode and vice versa.

3 Optimization Objectives

There are different possible objectives for scheduling problems. In the three field notation of scheduling problems by Graham, Lawler, Lenstra, and Rinnooy Kan [6] the objective is specified in the last field. During the last decades of scheduling research some objectives have been frequently chosen for research studies, like the makespan of a schedule, and there are many results for these objectives. The existence of such results is sometimes used as an argument to select one of the common objectives for a research study of practical relevance. However, unless the objective does not represent the goal of the system owner it is unlikely that the system owner is interested in the study. Therefore, it is important that the original objective of the system owner is described first. In order to use a common objective, we must show that every good solution using the common objective is also a good solution with respect to the original objective.

Many practical scheduling problems have more than one objective. For such problem, ideally the Pareto optima are detected. Since finding all or even a sufficient amount of Pareto optima is often very difficult and time consuming, multi-objective problems are often transformed into conventional single objective problems by turning one objective into a constraint, that is, a valid area for the corresponding value is fixed. This approach requires a final analysis to determine the deviation of this value in the solution from the target value since the problem solution may deliver a value anywhere in the valid area. This is particularly important if the transformed objective has a higher priority than the addressed objective.

In our IaaS example like in most commercial scenarios, the owner has the primary objective to increase his profit. The use of money and the corresponding evaluation of expenses is a practical approach to transform multi-objective optimization problems into single objective problems. Unfortunately, the single objective problem is usually too unspecific and too complex to be addressed by common optimization approaches without introducing additional assumptions. In our example, we assume that the owner has set a fixed hourly fee for every high priority job (interactive or user-facing) and a (lower) fixed hourly fee for every low priority job (batch). This approach does not correspond to the spot market concept of Amazon but remember that it is our goal to find an alternative to the spot market. Further, we assume that the total number of installed machines and the expenses for housing and personal are fixed and can be ignored for the purpose of optimization. The system owner can influence energy costs by migrating low priority virtual resources and shutting down idle physi-

cal resources or activating physical resources that are inactive. We have several objectives:

- We do not want to reject any high priority request unless all physical resources are busy executing high priority requests.
- We want to achieve the target value of power consumption, that is, a target number of physical machines should be active unless more machines are required to execute high priority virtual resources or there are not enough low priority virtual resources.
- Similar to the spot market we allow that a service guarantee of a low priority virtual resource is violated but we want our resource management system to do its best to prevent such a violation if active physical resources are available.

Often it is difficult to translate the objectives of a system owner into a mathematical form that has already been used in more theory oriented publications. Since it is necessary to show the equivalence or at least the close relationship between both formulations, we must use special care when describing such translation.

4 Algorithm for the IaaS Example

In this section we describe an online allocation algorithm for our IaaS scenario. This algorithm is later used to explain an evaluation approach.

In our IaaS scenario, we do not allow to request on-demand virtual resources for a future starting time. In this case customers must select a long term lease. These long term leases, for instance for basic interactive resources, are not considered in this section since they have the highest priority and are not allocated online. Our online algorithm Allocation in Fig. 1 describes the main steps of handling the request for a virtual resource R. The algorithm does not distinguish between flexible interactive virtual resources and user-facing virtual resources but combines them into on-demand virtual resources. On-demand virtual resources are allocated to customers one instance period Δ at a time, that is, on-demand allocations have a fixed length determined by the provider. There is a fixed price for each instance period of an on-demand allocation. Any on-demand allocation that has been started will be extended on request of the customer. The extension request must be received time δ before the end of the instance period of the current allocation. The provider may either decide to use an explicit extension or a default extension. In the latter case, any allocation is automatically extended for another instance period unless the extension is canceled at least time δ before the end of the current instance period. Therefore, there is only a single successor of a current on-demand allocation and this request must be received on time (at most time Δ before the completion of the current allocation). This property is tested with procedure *successor()*. The successor request is preferably allocated to the same physical resource as the current allocation but migration at the beginning of the new allocation is possible for the reason of shutting down processors (procedure *allocate_successor()*).

A new on-demand allocation will be accepted provided there are some resources that are not occupied by on-demand allocations starting at the current time plus δ at the latest. This start-up delay produces a stretch factor of at most $(\Delta + \delta)/\Delta$. If all physical resources are occupied then a batch virtual resource must be terminated (procedure *batch_termination*). The new on-demand virtual resource will be allocated to a suitable physical resources considering goals regarding idle resources (procedure *allocated_new()*).

Batch virtual resources can be requested for any integer multiple of an instance period up to a maximum value, say $k\Delta$. They can be preempted and migrated at any time. Batch virtual resources are charged a low rate per instance hour. The rate may be determined by bidding like in Amazon's spot market. There may be common deadlines for batch virtual resource allocations, for instance, every day at noon. Whenever a batch virtual resource is submitted it is assigned the earliest deadline such that its stretch factor exceeds a minimal value f_b for batch virtual resources. The provider selects f_b. A batch virtual resource will be *terminated* if there are not enough resources for on-demand virtual resources or if not all batch virtual resources cannot complete in time. Note that the expression *termination* does not necessarily mean that the batch virtual resource is stopped but that it will not be completed by its deadline. The provider selects the batch virtual resource that will be terminated. If a batch virtual resource is terminated then the customer is charged for the completed instance periods. If a batch virtual resource is completed by its deadline then the customer has to pay an additional charge for each instance period the batch virtual resource was running.

In case of a shortage of physical resources due to an unpredictable machine failure batch virtual resources have the lowest priority and are terminated first. If the number of physical resources is not sufficient to execute all on-demand virtual resources then the provider has to terminate some on-demand virtual resources and accept a penalty. For this selection procedure, the provider must also determine a policy.

5 Evaluation

Every paper on algorithms requires an evaluation of the new algorithm. Ideally such evaluation covers all valid problem instances, also called the *problem space*, and additionally considers the frequency of occurrence for each such problem instance. On the one hand, it is very difficult for many parallel job scheduling problems on real machines to provide an easy characterization of the problem space that formally separates it from all invalid problem instances due to the large number of constraints in such problems. On the other hand, an evaluation only requires such separation if invalid (or very rare) problem instances determine the outcome of the evaluation. The handling of the problem space is an important property of an evaluation approach.

In general, there are three major approaches for the evaluation of job scheduling algorithms for parallel processors:

Algorithm Allocation (Request R)

```
if (there is an on-demand virtual resource R' with successor(R')=R) {
    accept R;
    allocate_successor(R');
    start R immediately after completion of R'; }
else if (R is a new on-demand virtual resource) {
    if (all physical machines are occupied with on-demand resources) {
        reject R; }
    else {
        if (there is no idle physical machine) {
            batch_termination; }
        accept R;
        allocate_new(R);
        start R as soon as possible; } }
else {
    if (there is a valid batch schedule) {
        accept R; }
    else {
        reject R; } }
reschedule_batch;
```

Fig. 1. Algorithm for Acceptance of an Allocation Request R

- Theoretical analysis
- Execution on a real machine
- Simulation

Each of these approaches has advantages and disadvantages that must be taken into account when deciding how to evaluate an algorithm.

5.1 Theoretical Analysis

When discussing the theoretical analysis approach we distinguish between *easy* and *difficult* problems. For an easy problem, there is an algorithm with polynomial time complexity that always finds an optimal solution in the problem space if such solution exists. Here, theoretical analysis covers the whole problem space and is well suited.

For difficult problems, a proof of intractability rarely provides any benefit in practice with the exception of stating that there is no further need to look for an optimal algorithm. Therefore, many studies produce algorithms with performance guarantees like approximation or competitive factors for these problems. These guarantees are upper performance bounds (for minimization problems), that is, they consider specific worst case problem instances. For all other problem instances, we only know that the deviation from the optimum does not exceed this guarantee. Since the guarantee is often too large to be acceptable in a real life situation, the performance guarantee is seldom beneficial in practice.

The value of this information is further reduced if the performance guarantee is not tight or if it is determined by an invalid or very unlikely problem instance. Therefore, studies presenting new performance guarantees must also answer the following questions:

- Is the determined worst case performance guarantee also acceptable as an average performance deviation in practice?
- Does any problem instance that determines the performance guarantee has a high likelihood to occur in practice?

Many practitioners doubt the benefit of a theoretical evaluation for practical job scheduling problems. The already mentioned EASY backfilling is frequently used as a prominent example. This approach is applied in many parallel processing systems although it has a very bad performance guarantee. We still believe that theory also has similar benefits in the field of job scheduling as, for instance, for numerical simulation problems. In Section 6 we will give an example to show that theoretical analysis that can be helpful in practice.

5.2 Execution on a Real Machine

The performance of a newly designed algorithm can be evaluated by testing it in the field on a real system. In general this approach has the potential to consider constraints even if we have forgotten or neglected them when developing our algorithm. But this benefit does not necessarily hold when using a small test system due to a possible lack of scalability. Moreover, it is often very difficult to generate a real workload during a test without real customers. Testing on the target system with real customers avoids these disadvantages but it can only be used when the management system is mature enough to guarantee that customers are not alienated and no system problems are generated. Unfortunately, a test in the field is usually very expensive particularly if a large production system is involved. Therefore, it will normally not be used for the early stages of algorithmic evaluation although the ultimate test must occur on a real system. Most likely due to this effort, there are very few publications that report on algorithm evaluation on real systems.

But real systems are used to extract real workload data. Some of these workload data are made publicly available by storing them in repositories like the parallel workload archive for parallel computers[4]. Then they can be used directly or indirectly for an evaluation with the help of simulation studies.

5.3 Simulation

The majority of publications on job scheduling for parallel processing includes some simulation study. In general, a simulation study is comparable with an experiment in natural science since we generate an artificial environment to answer a research question. Since there is a much longer experimental tradition in

[4] www.cs.huji.ac.il/labs/parallel/workload

natural science than in computer science it is not surprising that no generally accepted approach for experiments in computer science has been established yet. The lack of such approach can be observed in many publications of simulation studies. It would be beneficial to borrow from the concepts of experiments in physics and chemistry. Since an experiment is designed to answer a research question this research question must be clearly formulated. Moreover, an experiment is only useful if it can be verified by other researchers. Therefore, it is necessary to describe all parts of the experiment in sufficient details to enable such verification.

The simulation experiment is based on a model of the real system. This model must consider all relevant constraints, see Section 2. While it is straight forward to include technical and organizational constraints, usage constraints are a more difficult challenge. As already stated in the beginning of this section we must sufficiently cover the problem space. Since each problem instance requires a separate simulation, a large number of simulations and a corresponding large number of input instances are necessary to guarantee such coverage.

Some publications use randomly generated data. While this approach is easy to realize it is not clear that random data can guarantee the required coverage. Therefore, every publication using random data must explicitly show that this condition is observed. Such proof is missing in many publications.

Alternatively, researcher often use real workload data from the already mentioned repositories. Although this approach seems to implicitly guarantee coverage of the problem space it also has some disadvantages. First of all, the number of existing workload traces in accessible repositories may not be sufficient to execute the required number of experiments to obtain meaningful results. Also trace data strongly depend on the environment in which they were recorded. Then we must determine whether a transfer to another environment is possible. For instance, fewer management conflicts will occur if the workload trace is recorded on a smaller system than the simulated system. If necessary then the simulated system must be adapted to the workload trace while still considering the other constraints. Moreover, a simulation with trace data is always history based and therefore not well suited to evaluate new algorithms if the environment allows strong interactions with participants. Such interactions often occur in market economies with a high volatility of demand or supply as new resource management algorithms may increase the supply of resources leading to a change in the demand of resources. For instance, the submission pattern of a user may change if the user knows that more resources are available. Therefore, a simulation study with trace data must always address the issue of interaction between system and providers of input data.

Due to the lack of trace data, some researchers use workload models like, for instance, the one proposed by Lublin and Feitelson [12] for parallel computers. These workload models are verified with workload traces and usually can be adapted to the simulation system. To support verification of the simulation study, the details of the adaptation must be part of the description of the simulation experiment. But since all workload models known to us are static models,

interactions with customers are not considered. In our view it is one of the key challenges in the area of job scheduling for parallel processing to develop workload models that incorporate a feedback component that changes the workload depending on the result of the resource management to imitate interaction with the participants.

6 Evaluation in the IaaS Example

In our example, we select the theoretical evaluation as at least some parts of the algorithm can be shown to be optimal. In this situation, a theoretical evaluation is beneficial since the problem space is obviously covered and better results are not possible for these parts, see Section 5.1. First, we must describe our problem in a way that is suitable for this kind of evaluation. We focus on on-demand virtual resources since these requests have a higher priority than batch virtual resources. The allocation of on-demand virtual resources can be described as a scheduling problem of jobs with unit processing time on parallel identical machines. Since we do not know which jobs will be extended we assume that independent jobs are submitted over time, that is, we have a classical online scheduling problem. Since we may only migrate an on-demand allocation before its start, we do not allow preemption in our problem. Due to the limitation of our stretch factor, the deadline of a job cannot exceed its submission time plus the stretch factor $1 + \delta < 2$ as we set $\Delta = 1$. We want to maximize the busy time of the physical resources, that is, we want to minimize the total idle time of these resources. The minimization of the total idle time due to on-demand virtual resources is equivalent to the minimization of the total number of jobs that cannot complete before or at their deadlines. Using the common notation in theoretical scheduling we have the objective function $\sum U_j$, see Pinedo [15]. Therefore, we can express our problem as $P_m|p_j = 1, r_{j,\text{online}}|\sum U_j$. Goldman et al. [4] address this problem on a single resource and show an upper bound of 0.5 for the competitive factor, that is, there are problem instances such that no deterministic online algorithm can finish more than half the number of jobs than an algorithm with complete knowledge of the submission sequence can complete. This upper bound is achieved by a simple greedy algorithm that accepts every job that will complete in time. Goldwasser [5] improves this result for slack factors $f \geq 2$. Kim and Chwa [9] show that the results of Goldman et al. and Goldwasser also hold for parallel identical machines. However, a close look at the proof of Goldman et al. shows that the proof uses a condition that may not hold in our problem. To discuss the difference between the general problem and our case in detail, we first introduce some notation. We say that job J_i with processing time $p_i = 1$ is submitted at release date r_i and has the deadline $d_i = r_i + 1 + \delta$. Goldman's proof requires that we have $d_i < d_j$ for two jobs J_i and J_j with $r_i > r_j$. This condition clearly does not hold for our problem. Remember that the deadline is not determined by the customer but by the provider. We can assume that the successor of an on-demand virtual allocation is submitted time δ

before the completion of the successor resulting in the above mentioned deadline as well. This special form of the general problem is addressed in Theorem 1.

Theorem 1. *In a system with parallel identical machines a greedy approach guarantees the minimum idleness if all jobs have the same processing time p and* $d_i \leq d_j$ *holds for any two jobs* J_i *and* J_j *with* $r_i \leq r_j$.

Proof. Consider an arbitrary non-preemptive schedule S with two jobs J_i and J_j such that $r_i \leq r_j$ and $c_i > c_j$ holds with c_i and c_j being the completion times of jobs J_i and J_j in schedule S, respectively. Then we simply exchange the positions of both jobs in the schedule. Clearly, job J_i can be started at time $c_j - p \geq r_j \geq r_i$. Similarly, job J_j will complete in time as $d_j \geq d_i \geq c_i$ holds. We apply this job exchange until all jobs start in the order of their release dates. Next, we transform the resulting schedule into a new schedule by applying greedy allocation to the jobs in the order of their completion times. Note that this transformation cannot increase the completion time of any job. Therefore, we only need to consider non-delay schedules in which the order of completion times corresponds to the order of release dates. For any such schedule S let $(c_1, c_2, \ldots)$ be the sequence of ordered completion times while $(c_1^{greedy}, c_2^{greedy}, \ldots)$ is the sequence of the ordered completion times of schedule S_{greedy} generated by the greedy approach. Due to greedy acceptance and equal processing times for all jobs, $c_i \geq c_i^{greedy}$ holds for any i if there are at least i jobs in both schedules. Therefore, no schedule is possible with more jobs than S_{greedy} as otherwise greedy acceptance does not reject the additional jobs resulting in a contradiction. As all jobs have the same processing time, S_{greedy} is optimal.

Since batch virtual resources are terminated if there are not enough physical resources for on-demand virtual resources our algorithm always produces an optimal allocation for on-demand virtual resources even if some new requests must be rejected due to a lack of physical resources. Remember that an extension of a current on-demand allocation will only be rejected in case of a machine failure.

Next, we address requests for batch virtual resources. In Algorithm Allocation, see Fig. 1, we must determine whether there is a valid schedule, that is, a schedule that completes all allocations before their deadlines. This test is necessary when a new batch virtual resource is submitted and if we must decide which batch virtual resource must be terminated. Contrary to on-demand virtual resources, we allow preemption for batch virtual resources. There are also some theoretical results for preemptive online schedule with similar objectives as the minimization of $\sum U_j$. Baruah and Haritsa [1] address preemptive online scheduling with the stretch metric on a single machine. They use the so called effective processor utilization (EPU). and are the first to present an algorithm that guarantees an EPU of $(f-1)/f$ with stretch factor f. DasGupta and Palis [3] show that the ratio $(f-1)/f$ is an upper bound for total utilization in the parallel identical machine environment if no migration of jobs is allowed. This bound is guaranteed by a greedy algorithm that accepts every job

provided that no deadline of any accepted job is violated. Again these results seem to indicate that it is not possible to guarantee an optimal result with a polynomial time algorithm. But our specific question whether there is a valid schedule for a given set of jobs is not an online problem since the number of jobs does not change while looking for an answer to the question. Moreover, we allow migration contrary to DasGupta and Palis [3]. Let us first assume that our batch virtual resources may have different deadlines. Then we look at the problem from the reverse direction by transforming deadlines into release dates and obtain the problem $P_m|r_j, \mathrm{prmp}|C_{\max}$. The reverse problem does not have any deadlines as all jobs are already available at time 0. This approach also allows us to consider physical resources occupied by on-demand virtual resources as we interpret these resources as additional jobs that will start at their release date and complete at the target deadline. The problem $P_m|r_j, \mathrm{prmp}|C_{\max}$ can be solved with a simple longest remaining processing time (LRPT) approach, see Pinedo [15]. If all batch virtual resources have the same deadline then we can also apply a forward approach with additional machines becoming available as soon as they are not occupied any more by on-demand virtual resources.

However, we cannot find an optimal algorithm with polynomial time complexity to determine which batch virtual resources to terminate in case of a lack of physical machines. Even if we do not consider the online character of the problem resulting from future submission of on-demand virtual resources, the problem can be reduced to the partition problem. Therefore, the provider must use some heuristic to determine which batch virtual resources to terminate.

7 Conclusion

In this paper, we tried to define some rules that may help to bridge the gap between algorithmic developers and practitioners in the area of job scheduling for parallel processing. These rules are not supposed to be strict laws that must be observed by every paper but rather be guidelines that allow some degree of flexibility. While every researcher can interpret the rules according to his specific problem it is also his responsibility to explain these interpretations sufficiently well to prevent misunderstanding between researchers and practitioners. Based on experience with numerous research papers in this area, we also feel that it is necessary to establish an approach for simulation experiments that is related to the approach used in experiments in natural sciences. Moreover, we hope that new workload models will be developed that help to consider interactions between users and the system as such interactions will take place more frequently due to the increasing use of market concepts in parallel processing.

We have used a simple IaaS scenario to provide an example how we feel that these rules should be applied in practice. This example can also be considered as a statement that in job scheduling for parallel processing an evaluation with theory methods may also be helpful in practice. It extends the well known concept to use the flexibility of long running jobs with low priority to improve data center efficiency. Contrary to Amazon's spot instances, we suggest that the flexibility

of these jobs does not require an immediate termination of a job in case of a resource shortage. Instead we suggest that a delay of the job may be beneficial for provider and customer. In addition, we suggest a modified pricing model that incorporates a surcharge if the job is completed in time. This way we replace the strict model of service guarantee for these jobs with a more flexible concept.

Acknowledgment

The author would like to thank Carsten Franke from ABB, Switzerland for providing information on current power management problems and approaches in data centers.

References

1. S.K. Baruah and J.R. Haritsa. Scheduling for overload in real-time systems. *IEEE Trans. Computers*, 46(9):1034–1039, 1997.
2. W. Cirne and E. Frachtenberg. Web-scale job scheduling. In W. Cirne, N. Desai, E. Frachtenberg, and U. Schwiegelshohn, editors, *Proceedings of the sixteenth international workshop job scheduling strategies for parallel processing*, volume 7698 of *Lecture Notes in Computer Science*, pages 1 – 15. Springer Berlin / Heidelberg, 2012.
3. B. DasGupta and M.A. Palis. Online real-time preemptive scheduling of jobs with deadlines on multiple machines. *Journal of Scheduling*, 4(6):297–312, 2001.
4. S.A. Goldman, J. Parwatikar, and S. Suri. Online scheduling with hard deadlines. *Journal of Algorithms*, 34(2):370 – 389, 2000.
5. M.H. Goldwasser. Patience is a virtue: the effect of slack on competitiveness for admission control. In *Proceedings of the tenth annual ACM-SIAM symposium on discrete algorithms*, SODA, pages 396–405, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.
6. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnoy Kan. Optimization and approximation in deterministic, sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287 – 326, 1979.
7. M. Ibrahim, S. Gondipalli, S. Bhopte, B. Sammakia, B. Murray, K. Ghose, M.K. Iyengar, and R. Schmidt. Numerical modeling approach to dynamic data center cooling. In *Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2010 12th IEEE Intersociety Conference on*, pages 1–7, 2010.
8. J.M. Kaplan, W. Forrest, and N. Kindler. Revolutionizing data center energy efficiency. Technical report, McKinsey & Company, 2008.
9. J.H. Kim and K.Y. Chwa. On-line deadline scheduling on multiple resources. In *Proceedings of the 7th annual international conference on computing and combinatorics*, COCOON, pages 443–452, London, UK, UK, 2001. Springer-Verlag.
10. C.B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Proceedings of the tenth international workshop on job scheduling strategies for parallel processing*, volume 3277 of *Lecture Notes in Computer Science (LNCS)*, pages 253–263. Springer Berlin/Heidelberg, 2005.

11. D. Lifka. The ANL/IBM SP scheduling system. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of the first international workshop on job scheduling strategies for parallel processing*, volume 949 of *Lecture Notes in Computer Science (LNCS)*, pages 295–303. Springer Berlin/Heidelberg, 1995.
12. U. Lublin and D.G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105 – 1122, 2003.
13. J. Mars, L. Tang, R. Hundt, K. Skadron, and M.L. Souffa. Bubble-up: Invreasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, New York, NY, USA, 2011.
14. A.K. Mishra, J.L. Hellerstein, W. Cirne, and C.R. Das. Towards characterizing cloud backend workloads: Insights from google computer clusters. *SIGMETRICS Performance Evaluation Review*, 37(4):34 – 41, 2010.
15. M.L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer Science+Business Media, forth edition, 2010.
16. R. Strong, J. Mudigonda, J.C. Mogul, N. Binkert, and D. Tullsen. Fast switching of threads between cores. *ACM SIGOPS Operating Systems Review*, 43(2):35–45, 2009.