# Influence of Dynamic Think Times on Parallel Job Scheduler Performances in Generative Simulations

Stephan Schlagkamp

Robotics Research Institute, TU Dortmund University,
Dortmund, Germany
stephan.schlagkamp@udo.edu

**Abstract.** The performance of parallel schedulers is a crucial factor in the efficiency of high performance computing environments. Scheduler designs for practical application focusing on improving certain metrics can only be achieved, if they are evaluated in realistic testing environments. Since real users submit jobs to their respective system, special attention needs to be spent on their job submission behavior and the causes of that behavior. In this work, we investigate the impact of dynamic user behavior on parallel computing performances and analyze the significance of *feedback* between system performance and future user behavior. Therefore, we present a user-based dynamic workload model for generative simulations, which we use to analyze the impact of dynamically changing think times on simulations. We run several such simulations with widely known scheduling techniques *FCFS* and *EASY*, providing first insights on the influence of our approach on scheduling performances. Additionally, we analyze the performances by means of different metrics allowing a discussion on user satisfying performance measures.

**Keywords:** Workload, Generative Simulation, User Behavior, Feedback

## 1 Introduction and Related Work

So far, a common technique to compare performances of different schedulers is achieved by simulations using previously recorded workload traces. There are many studies on analyzing properties of workloads, e.g., [9], resampling workload, e.g., [14], or prediction of future workload, e.g. [2]. Figure 1 depicts this situation: We use a previously recorded workload trace to measure the performance of a certain scheduler.

According to Schwiegelshohn, this technique does not suffice to gain practical performance measures. Schwiegelshohn describes a gap between scheduling in theory and its practical application [10]. According to him, there is a need to "prevent misunderstanding between researchers and practitioners", e.g., by comprehensible interpretations and conclusions from analyses. Additionally, he

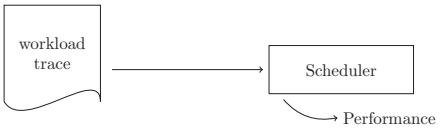


**Fig. 1.** Measuring performance with workload traces.

describes the necessity of workload models including a simulation of interaction of users and the system due to the spreading of the parallel computing concept. In this work, we address both discussed aspects: We develop a simulation framework for generative simulations of users interacting with a parallel computing system giving the opportunity to test schedulers in a real world simulation environment.

We present and analyze the results of a generative simulation and argue why these simulations must be of dynamic fashion. Since the process of users submitting jobs to a computing environment and receiving a response once their job was computed is based on user behavior. Users may react to sparse resources changing the workload or submit times, which are then faced by a certain scheduler. Testing scheduler performances by using earlier recorded workload traces suffices from a lack of these interactive effects. Therefore, we model a dynamic and interactive simulation environment focusing on *user feedback* [4]. Figure 2 depicts this approach: The workload is not determined beforehand but the results of the scheduler influence the user in his future behavior. The performance is an outcome of this generative process. Regarding this idea, each recorded workload trace is only one instantiation of a dynamic interaction process.

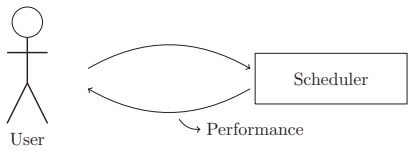


**Fig. 2.** Performance evaluation including feedback.

We can think of many different forms of feedback between users and a parallel computing environment:

- People could start their daily work earlier or finish later, if the system does not offer satisfiable performances.
- Contrary, they could begin their work later, or finish earlier, if the responsiveness of the system is *good*.
- In a system with poor performances, people could tend to work on weekends, to find it less utilized, assuming that they prefer working on weekdays.

– Users could tend to change the characteristics of jobs they submit. Job parameters (size, length, etc.) may be adapted to get results faster or resources are used more efficiently.
– In case the system is advanced by further resources, job characteristics might be adjusted accordingly.
– Runtime estimates can have a major influence on scheduling performances [12]. We can also think of them being *tuned* by users, to receive more satisfying results.

So far, little or none dynamic simulations were conducted in the context of parallel scheduler evaluations. Although we discussed different possible forms of feedback, we want to focus on *think times*, which is the interval between response and submission of two consecutive jobs. This form of feedback is analyzed in different works, e.g., [4].

Feitelson describes the reaction of users to system performances as "a mystery" [6, p. 414]. The workload submitted by users and the system performance should meet in a *stable state*. A growing demand leads to poorer system performance (cf. Figure 3). This result can be obtained in a performance test with increasing workload. Nevertheless, the actual user reaction is an open question.

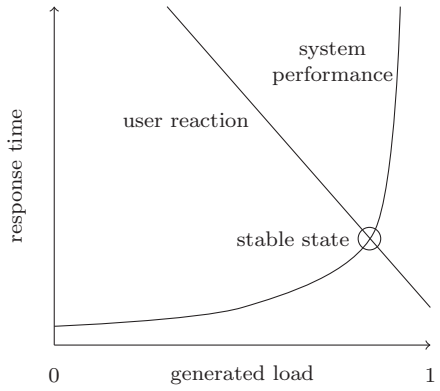


**Fig. 3.** Supply-and-demand curves crossing in stable state [6, p. 414].

Our simulations are based on the *teikoku Scheduling framework* [1] which is extended by a workload generation module simulating dynamic user behavior. Furthermore, the only feedback we model is dynamic think time. Future work can adapt this model to simulate further feedbacks. We measure the performance by four different metrics, which can all be interpreted as describing an impact on user satisfaction. Results from simulations as provided in this paper can be used to justify certain goals of optimization in applied scheduling strategies to make results more comprehensible for practitioners.

This work is structured as follows. In the next section, we give an overview of related work. Afterwards, we present the dynamic user model as the basis of the simulation and discuss different feedback functions in Section 2. In Section 3, we present the results of our simulation and discuss the influence of dynamic think times. We close this work with a conclusion in Section 4.

## 2 User Model and Feedback

To develop our simulation environment, we make two a priori assumptions. First, submission times and working habits are based an a weekly pattern, e.g., described in [6, p. 394]. Regularly, people work from Monday to Friday followed by the weekend from Saturday to Sunday. We assume that this observation holds for the work with HPC environments to some extend, as well. Second, users do not work all day long. We suppose that they start at a certain point in time and finish work later, according to different days of the week. These assumptions lead to the framework of our simulation. Given a weekly structure as the global frame of user behavior, our simulation can run for an adjustable number of $n_w$ weeks. Furthermore, we model $n_u$ individual users $u \in U$ in the set of all users $U$ participating in a simulation run. Users might be more or less active, as Feitelson and Shmueli have analyzed [11]. We take care of this fact by introducing the *activity ratio*

$$p_{a,u} \in [0,1], \tag{1}$$

which is the percentage chance of a user being active in a certain week.

After describing this main structure of the simulation, we can now focus on attributes describing the submission behavior of a single user. To model different activity at different days of the week, we introduce a distribution describing this activity

$$\begin{aligned} p_{d,u} \in [0,1] \quad & \forall d \in D = \{mon, tue, \ldots, sun\}, \forall u \in U, \\ & \sum_{d \in D} p_{d,u} = 1 \quad \forall u \in U. \end{aligned} \tag{2}$$

Every user has a certain point in time to start and to end his or her day. Additionally, we introduce variables

$$t_{b,u} \in [0, 86400], \tag{3}$$

$$t_{e,u} \in [0, 86400] \quad \forall u \in U, \tag{4}$$

$$t_{b,u} < t_{e,u}$$

describing an individual start and end of their working day in seconds. In case the submit time of a new job is not between $t_{b,u}$ and $t_{e,u}$, it is delayed until the next day begins. Additionally, we focus on job characteristics. We restrict the number of processors per job to powers of two, as other numbers of requested processors are fairly uncommon [4]. The number of requested processors of a

job and its running time are not correlated over different systems [4]. However, we assume that jobs in different applications tend to have same characteristics regarding their sizes and running times, or that the same user submits jobs of the same type. Therefore, we model a correlation of job characteristics according to each user. For each user $u$, we give the probability of choosing a certain number of processors for his job

$$p_{u,m_j} \in [0,1],$$
$$\sum_{m_j} p_{u,m_j} = 1 \quad \forall u \in U, m_j \in \{2^i \mid i \in N\}. \tag{5}$$

The normal distributions of running times for a given number of processors

$$\mu_{u,m_j}, \tag{6}$$
$$\sigma_{u,m_j} \quad \forall u \in U, m_j \in \{2^i \mid i \in N\}, \tag{7}$$

are set for each user respectively.

Keeping the model simple, we introduce a linear think time function. Two variables represent each user's specific think time behavior. Variables $tt_{u,m}$ and $tt_{u,b}$ are used in the the linear function

$$tt_u(r_j) = tt_{u,m} \cdot r_{j-1} + tt_{u,b}, \tag{8}$$

giving the think time according to the response time of the last job $j_{i-1}$ submitted by user $u$.

Note that we do not model user sessions in detail, due to the difficulty of extracting session information from workload traces [13]. In this work, our goal is to describe effects caused by dynamic think times. Therefore, we restrict our user models from users submitting jobs in parallel, while still waiting for their results. Figure 4 depicts an overview of the described process.

Now we can analyze some situations for which we artificially set up a set of users in a certain system before we extract data from existing workload traces and run *realistic*, feedback-aware simulations. Although not all users might fit in the proposed model due to different working habits, e.g., there might be people starting their work one day and working over midnight to finish next day, we hope that the simulation is a first step towards user-aware simulation.

We arbitrarily choose four workload traces to learn parameters $tt_{u,m}$, and $tt_{u,b}$. Traces LANL CM5, KTH-SP2, OSC Cluster, HPC2N, ANL Intrepid, and SDSC SP2 range from 213-437 users with 28,489-527,371 jobs, and 100-163,840 cores [5]. Figure 5 depicts a plot of the think times in the chosen traces. Only jobs $j_i$ having a subsequent job $j_{i+1}$ of the same user are considered. They must not overlap, i.e., the beginning of $j_{i+1}$ must be after $j_i$ finished. Furthermore, only think times of less than 8 hours are considered: $0 < s_{i+1} - r_i < 28,800$ seconds, with submit time $s_{i+1}$ of job $j_{i+1}$ and response time $r_i$ of job $i$. Fitting a linear function to the provided data, we receive $tt_{u,m} = 0.4826$, and $tt_{u,b} = 1779$, when least-squares is applied.

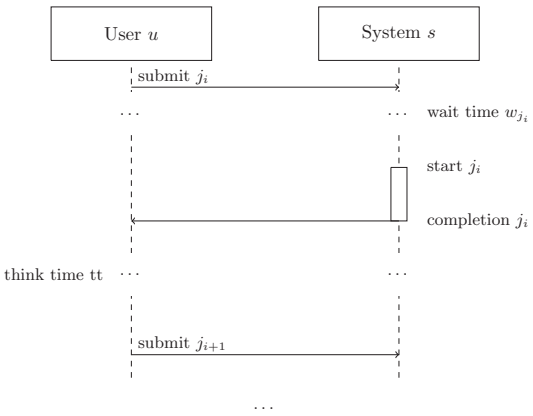


**Fig. 4.** Job submission workflow of user $u$.

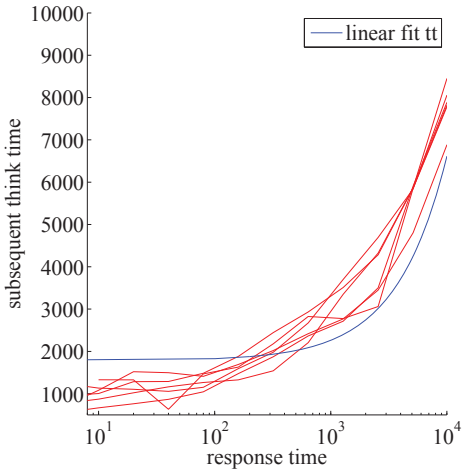


**Fig. 5.** Linear fit of think times in different workload traces.

## 3 Generative Simulations

We are interested in the effect on the following different measurements of performances (cf. [3]). We consider the following four metrics.

- *Average Response Time (ART)*: The sum of all response times (time passing from job submission to receiving of the result) divided by the number of jobs

$$\mathrm{ART} = \frac{\sum_{j \in J} r_j}{|J|}.$$

- *Average Weighted Response Time (AWRT)*: The response time of each job is weighted by its size and divided by the number of jobs submitted in total,

$$\mathrm{AWRT} = \frac{\sum_{j \in J} p_j \cdot m_j \cdot r_j}{\sum_{j \in J} m_j}.$$

- *Average Waiting Time (AWT)*: All waiting times (time passing from job submission to actual processing) are normalized by the number of jobs

$$\mathrm{AWT} = \frac{\sum_{j \in J} w_j}{|J|}.$$

- *Average Slowdown (ASD)*: The slowdown is defined as the response time normalized by the running time. We sum up all slowdowns and take the average

$$\mathrm{ASD} = \frac{\sum_{j \in J} \frac{r_j}{p_j}}{|J|}.$$

We choose these metrics for the following reasons. Since we model the think time as a function of response time, we implicitly assume that different time measures of a job have a certain impact on users. The ART is the time a user has to wait in total for a result and is therefore considered. Furthermore, the AWRT takes the size of a job into consideration and might also be of interest for certain user behavior. Additionally, we focus on the waiting time. While AWT expresses the waiting times users face, SD expresses the proportional running time according to the processing time (and therefore indicates on the proportional waiting time).

The value of the makespan is not considered due to the following two reasons. First, weekly patterns with a break at weekends overturn the concept of the makespan minimizing the overall computation time. Second, it is not a measure for user satisfaction. In a parallel computing environment, users are not interested in the point in time when the last job among all jobs (possibly submitted in the future) finishes but when each of their own submitted jobs finish.

In this work, we choose two commonly known scheduling techniques for comparison.

- *First Come First Serve (FCFS)*: Using FCFS, jobs are computed in order of arrival at the system.

– *EASY backfilling (EASY)*[8]: If the running time of a job is known or at least an estimation is given, a queued job is preferred if enough machines are available at the current point in time and the job does not take longer then the so far scheduled jobs take.

In our simulations, EASY has an optimal working environment due to perfectly known running times at forehand. Clearly, this is an advantage for the performance of this scheduling technique. Future analyses may also consider mistakes in runtime estimations, as they are not necessarily of good quality in practice [7].

### 3.1 Artificial Simulations

In the artificial setup, we want to analyze potential differences and finding out how far the performance of scheduling strategies is influenced according to the previously discussed metrics. We therefore use the following user archetype: Each user works from 9 a.m. to 5 p.m. ($u_b = 32400$, $u_e = 61200$) and working time is evenly distributed among weekdays only, i.e.,

$$p_{d,u} = 0.2 \quad \forall d \in \{mon, tue, wed, thu, fri\}$$

and

$$p_{d,u} = 0 \quad \forall d \in \{sat, sun\}.$$

As we seek to find general insights on the influence of think time, we arbitrarily choose the correlation that larger jobs take longer, which can be found in some workloads [6, p. 378]. Therefore, we choose a Gaussian-like distribution for job sizes. Length distributions grow from $\mu_u(2) = 20$ seconds for two cores up to $\mu_u(1024) = 14400$ seconds for 1024 cores with standard deviation of $\sigma_u(m_j) = \frac{\mu_u(m_j)}{2}$. Future work may address further correlations. To simulate realistic user behavior, suitable values have to be extracted from workload traces or analyses of user behavior.

As we want to investigate whether there is a difference when linear think time is present, we compare it to constant behavior. Therefore, we choose the following three different think time functions:

– Constant: 20 minutes, $tt_{u,c_{20}}(r_j) = 1200$
– Constant: 120 minutes, $tt_{u,c_{120}}(r_j) = 61200$
– Constant: 240 minutes, $tt_{u,c240}(r_j) = 61200$
– Linear: linear think time model,
  $tt_{lin}(r_j) = m_u \cdot r_j + b_u, m_u = 1.051, b_u = 1361$ (cf. Section 2)

To simulate an interactive process, we generate ten users sampling and submitting jobs to a system $s$ of size $m_s$. Job sampling and submission is fulfilled according to the described parameters and one certain think time function. Each run simulates ten weeks. The number of cores present in the system $m_s$ decreases from 1024, 512, 256 to 128 to simulate different utilization scenarios. Whenever

a user samples a job, which is greater than the currently simulated system size, the job size is reduced to system size, i.e., $m_j \leftarrow \min\{m_s, m_j\}$. This means, the probability that a job needs all cores of the system increases to $\sum_{m_j \geq m_s} p_{m_j}$. Furthermore, each simulation is performed with schedulers FCFS and EASY (cf. Section 3).

Summarizing, one simulation run has the following attributes:

- System size $m_s \in \{1024, 512, 256, 128\}$
- Scheduler: FCFS or EASY
- Think time model: $tt_{c20}$, $tt_{c120}$, $tt_{c240}$, or $tt_{lin}$
- Simulation of ten weeks

All parameters describing a single user are summarized in Figure 6. To create

| Parameter | Value | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Activity ratio $p_a$ | 0.5 | | | | | | | | | | | |
| Start of day $t_b$ | 32400 | | | | | | | | | | | |
| End of day $t_e$ | 61200 | | | | | | | | | | | |
| Activity within week | mon | tue | wed | thu | fri | sat | sun | | | | | |
| | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0 | 0 | | | | | |
| Job attributes | Cores $m_j$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
| | Distribution $p_{m_j}$ | 0.0 | 0.0 | 0.05 | 0.05 | 0.1 | 0.15 | 0.3 | 0.15 | 0.1 | 0.05 | 0.05 |
| | $\mu_{m_j}$ | 0.0 | 20 | 38 | 75 | 150 | 300 | 600 | 1200 | 3600 | 7200 | 14400 |
| | $\sigma_{m_j}$ | 0 | 10 | 20 | 38 | 75 | 150 | 300 | 600 | 1200 | 3600 | 7200 |
| Think times | Parameter $tt$ | $c_{20}$ | $c_{120}$ | $c_{240}$ | lin | | | | | | | |
| | Think time $tt_{u,m}$ | 0.0 | 0.0 | 0.0 | 0.4826 | | | | | | | |
| | Think time $tt_{u,b}$ | 1200.0 | 7200.0 | 14400.0 | 1779.0 | | | | | | | |

**Fig. 6.** Basic parameters defining all users in our artificial simulation setup.

convincing data, we repeat each simulation configuration for 100 times.

**Influence of Different Think Time Functions** We cannot distinguish whether the length of think times are caused by the working habits and type of work performed by users or if there is some psychological reason for such behavior. However, these experiments will give arguments on positive or negative effects on the metrics considered. Furthermore, we can analyze which metrics are influenced more than others, which might allow us to draw conclusions on the needs of users in parallel computing environments. Comparing the four different think time models we also have to take the processed workload into account. Taking the workload into consideration allows clearer comparisons as of the nature of the analyzed problem: in a less utilized system the chance of better scheduling results according to certain metrics might be easier. We measure the workload in *processor hours*, which describe the amount of workload processed on the system. All running times of all cores are summed up.

The results of all simulations are presented in Figure 7 and Figure 8. Each row of box plot charts represents one system size (128, 256, 512, or 1024 cores). Within each row, the values for the four metrics, as well as the processed workload are depicted for all four different think time functions named c20, c120, c240, and lin. A single box plot is the graphical representation of all metric or workload values gained at the 100 simulation runs. At the first glance, we can see that the processed workload decreases for increasing constant think times. The longer a user waits between job submissions, the less workload must be handled by the system in the simulated ten weeks interval.

For both schedulers, FCFS and EASY, the metric values decrease according to the decrease in workload. We find this pattern in all charts, except for the ASD value when EASY is applied. For 512 and 1024 cores, think times of 20 and 120 minutes do not influence ASD *significantly*, i.e., for 512 cores the box plots look fairly equal, while for 1024 cores the ASD is worse for the median, as well as for the first and third quartile. A reason for that might be that our simulated job submissions are not influencing the performance for these two special settings, since we observe the decrease in all other simulations.

Interestingly, the linear think time model influences the metrics significantly, as the following detailed analyses show. We compare the results gained for both schedulers for the initial setup of 1024 cores, and the most reduced simulated system of 128 cores. We use Figure 9 - Figure 12 to analyze differences in performances. In each Figure, the 2.5 percentile $Q_{2.5\%}$, median, and 97.5 percentile $Q_{97.5\%}$ depict the performance of different runs of simulations for two different think time models. Additionally, we highlight the difference in percentage between both simulations. We use three different ways to emphasize this difference in percentage: In case the first think time model is better than the second, the percentage value is bold. We use italic writing in case the difference is less than the processed workload, e.g., 2% more workload is processed but the value for a certain metric is only larger by 1%. Otherwise, the value is not emphasized.

In a 1024 cores system, the linear think time model leads to better results compared to the constant model c120. In Figure 9, we compare lin to c120. Even though more workload is processed for lin (the median is better by 8.1%), all metric values are relatively better than the ones for c120. The median of AWRT is only higher by 0.7%, ART by 1.2%, AWT by 1.2%, and ASD by 1.4%. All quantiles are at least relatively better for lin compared to c120.

Similar results hold for the simulation of FCFS in a 128 cores system (cf. Figure 10). Again, we compare lin and c120. The linear model produces 0.3% more workload considering the median, but all four metrics are better between 12.1% and 4.4% for the linear workload generation. The same holds for $Q_{2.5\%}$ and $Q_{97.5\%}$ which vary between 0.4% to 11.2%. This example clearly shows, that at almost equal workload, linear think times outperform static think times. The slowdown seems to be influenced most by the workload FCFS has to handle compared to the influence on other metrics: while the difference of the median of lin compared to c120 in the 1024 cores simulation is close to the to the differences

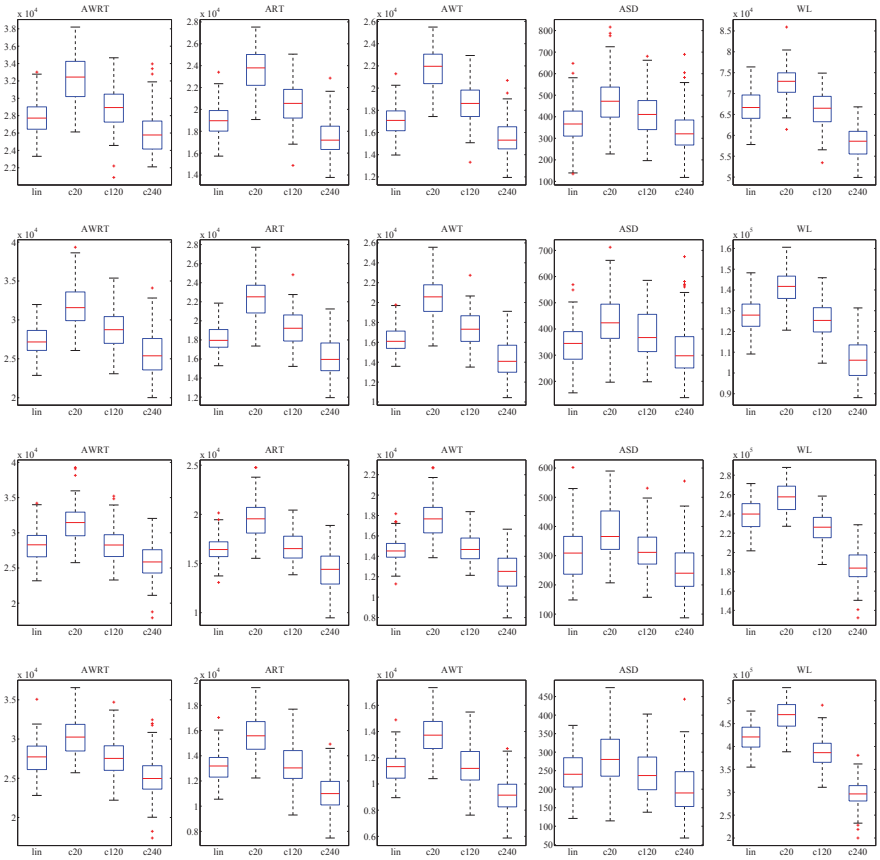


**Fig. 7.** Simulation Results of artificial simulation with 128, 256, 512, 1024 and FCFS scheduler.

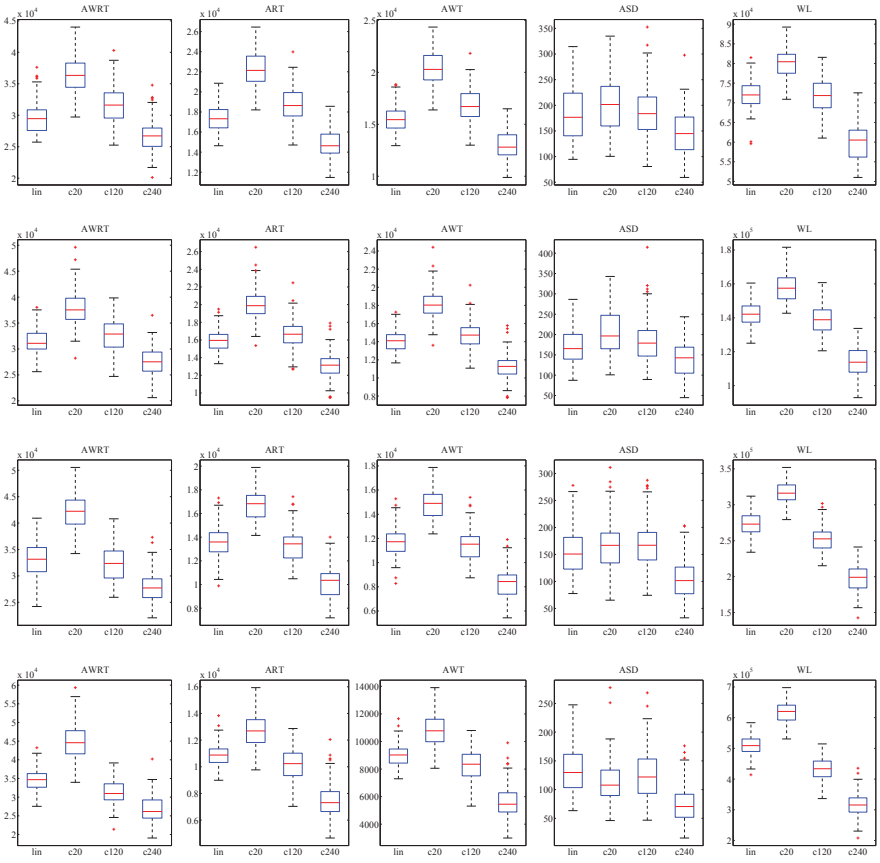


**Fig. 8.** Simulation Results of artificial simulation with 128, 256, 512, 1024 and EASY scheduler.

| FCFS | 1024 | $Q_{2.5\%}$ | Median | $Q_{97.5\%}$ |
|---|---|---|---|---|
| AWRT | lin | 23598.44 | 27720.71 | 31905.83 |
| | c120 | 23315.24 | 27532.03 | 33139.81 |
| | | *1.2%* | *0.7%* | **-3.9%** |
| ART | lin | 10985.66 | 13189.83 | 15842.24 |
| | c120 | 10588.10 | 13036.69 | 17367.84 |
| | | *3.6%* | *1.2%* | **-9.6%** |
| AWT | lin | 9278.85 | 11322.33 | 13828.88 |
| | c120 | 8855.86 | 11187.99 | 15280.66 |
| | | *4.6%* | *1.2%* | **-10.5%** |
| ASD | lin | 129.01 | 240.32 | 356.20 |
| | c120 | 141.30 | 236.85 | 376.36 |
| | | **-9.5%** | *1.4%* | **-5.7%** |
| WL | lin | 1305272612 | 1515061506 | 1694093600 |
| | c120 | 1200556728 | 1391623828 | 1625874940 |
| | | **8.0%** | **8.1%** | **4.0%** |

**Fig. 9.** Metric values for FCFS in artificial setup with 1024 cores.

| FCFS | 128 | $Q_{2.5\%}$ | Median | $Q_{97.5\%}$ |
|---|---|---|---|---|
| AWRT | lin | 23862.72 | 27720.17 | 31576.47 -8.2 |
| | c120 | 24569.47 | 28935.14 | 34166.52 |
| | | **-3.0%** | **-4.4%** | **-8.2%** |
| ART | lin | 16781.50 | 18964.58 | 22017.13 |
| | c120 | 16846.89 | 20547.47 | 24071.83 |
| | | **-0.4%** | **-8.3%** | **-9.3%** |
| AWT | lin | 15055.25 | 17086.67 | 19782.00 |
| | c120 | 15172.02 | 18618.06 | 22002.97 |
| | | **-0.8%** | **-9.0%** | **-11.2%** |
| ASD | lin | 209.74 | 366.45 | 581.06 |
| | c120 | 221.61 | 410.72 | 645.76 |
| | | **-5.7%** | **-12.1%** | **-11.1%** |
| WL | lin | 214225436 | 239972934 | 267924464 |
| | c120 | 206598240 | 239365340 | 267277068 |
| | | **3.6%** | **0.3%** | **0.2%** |

**Fig. 10.** Metric values FCFS in artificial setup with 128 cores.

of all other metrics, it is the value of greatest difference between lin and c120 for 128 cores.

We now compare the performance of EASY with 1024 cores to the performance if higher workload is generated in a 128 core scenario, and again choose the c120 simulations to compare both results (cf. Figure 11).

The processed workload in the 1024 cores simulation with static think time c120 is outperformed by 14.8% at median by lin. Except for AWT, with a $Q_{2.5\%}$ of 21.1%, all values of $Q_{2.5\%}$, median, and $Q_{97.5\%}$ are relatively better according to the linear think time respecting the difference in workload.

| EASY | 1024 | $Q_{2.5\%}$ | Median | $Q_{97.5\%}$ |
|---|---|---|---|---|
| AWRT | lin | 29084.08 | 34692.37 | 41745.11 |
| | c120 | 24694.65 | 30978.13 | 36791.09 |
| | | *15.1%* | *10.7%* | *11.9%* |
| ART | lin | 9187.50 | 10882.36 | 12750.43 |
| | c120 | 7247.76 | 10235.89 | 12481.95 |
| | | 21.1% | *5.9%* | *2.1%* |
| AWT | lin | 7463.42 | 9022.85 | 10761.17 |
| | c120 | 5564.28 | 8353.36 | 10486.81 |
| | | 25.4% | *7.4%* | *2.5%* |
| ASD | lin | 72.61 | 129.72 | 227.23 |
| | c120 | 63.89 | 121.96 | 223.41 |
| | | *12.0%* | *6.0%* | *1.7%* |
| WL | lin | 1574387124 | 1832448410 | 2063426668 |
| | c120 | 1326863512 | 1561360172 | 1808671904 |
| | | **15.7%** | **14.8%** | **12.3%** |

**Fig. 11.** Metric values for EASY in artificial setup with 1024 cores.

In Figure 12, all values are depicted for EASY in a 128 cores simulation. The processed workloads of lin and c120 are almost equal, with a median difference of 0.2%. Except for the $Q_{2.5\%}$ value of ASD, all values are at least relatively better.

While ASD was influenced almost best in the 1024 cores setup with difference of 6.0% at median (ART was best with 5.9%), we now observe the lowest difference of $-4.0\%$. All three values for lin are worse compared to the differences for the other metrics. We conclude, that ASD is most influenced by jobs submitted with linear think time in different system sizes.

Furthermore, by simply reusing a workload trace without dynamic think times, this significant difference in performance, from best difference to worst difference among all considered metrics, would not have been discovered.

For both scheduling techniques considered, an adaptive think time behavior leads to better performances. According to the four metrics considered, they perform better if we compare lin to constant models, in case equal processed

| EASY | 128 | $Q_{2.5\%}$ | Median | $Q_{97.5\%}$ |
|---|---|---|---|---|
| AWRT | lin | 25875.20 | 29462.72 | 36106.23 |
| | c120 | 25826.87 | 31617.68 | 37162.35 |
| | | *0.2%* | **-7.3%** | **-2.9%** |
| ART | lin | 14939.06 | 17321.88 | 20617.80 |
| | c120 | 15373.38 | 18638.35 | 21449.06 |
| | | **-2.9%** | **-7.6%** | **-4.0%** |
| AWT | lin | 110.62 | 176.66 | 306.30 |
| | c120 | 97.57 | 183.75 | 302.05 |
| | | **-2.2%** | **-8.2%** | **-4.2%** |
| ASD | lin | 110.62 | 176.66 | 306.30 |
| | c120 | 97.57 | 183.75 | 302.05 |
| | | 11.8% | **-4.0%** | *1.4%* |
| WL | lin | 237278984 | 259226294 | 285866200 |
| | c120 | 228554012 | 258636800 | 290397720 |
| | | **3.7%** | **0.2%** | **-1.6%** |

**Fig. 12.** Metric values for EASY in artificial setup with 128 cores.

workload sizes are taken into account. Although the difference in performances decreases in simulations with sparse resources, the statement still holds.

Several conclusions can be drawn towards user satisfaction regarding ASD. ASD is not minimized when sparse resources are met and a linear think time is applied. This value is influenced most in different setups. These observations suggesting further analyzes on this metric specifically.

**Performances of Scheduling Strategies** We contrast the performances of FCFS and EASY with the linear think time model. In Figure 13, and Figure 14, we compare the results for 1024 and 128 cores, respectively.

EASY outperforms FCFS for ART, AWT, and ASD up to more than 85%, for both, 1024, and 128 cores. Interestingly, we do not observe this for AWRT. Seeing the differences in workload, performances for EASY and FCFS are almost relatively equal. Dynamic think times seem to influence the scheduling techniques significantly, as EASY does not outperform FCFS for this certain metric, even though small jobs are favored by EASY and optimal job length estimates are provided.

## 4 Conclusion

We introduced a generative simulation environment to analyze feedback effects. We modeled user reactions to system performances in form of dynamic think times. In several analyses, we showed that this dynamic reaction outperforms static behavior. The results must be verified in further studies and more work of this type is important to find arguments for or against certain metrics to rate user satisfaction. Even psychological research should focus on the presented aspects to understand human behavior in HPC environments more deeply.

| EASY/ FCFS | 1024 | $Q_{2.5\%}$ | Median | $Q_{97.5\%}$ |
|---|---|---|---|---|
| AWRT | F.lin | 29084.08 | 34692.37 | 41745.11 |
| | E.lin | 23598.44 | 27720.71 | 31905.83 |
| | | 18.9% | 20.1% | 23.6% |
| ART | F.lin | 9187.50 | 10882.36 | 12750.43 |
| | E.lin | 10985.66 | 13189.83 | 15842.24 |
| | | **-19.6%** | **-21.2%** | **-24.2%** |
| AWT | F.lin | 7463.42 | 9022.85 | 10761.17 |
| | E.lin | 9278.85 | 11322.33 | 13828.88 |
| | | **-24.3%** | **-25.5%** | **-28.5%** |
| ASD | F.lin | 72.61 | 129.72 | 227.23 |
| | E.lin | 129.01 | 240.32 | 356.20 |
| | | **-77.7%** | **-85.3%** | **-56.8%** |
| WL | F.lin | 1574387124 | 1832448410 | 2063426668 |
| | E.lin | 1305272612 | 1515061506 | 1694093600 |
| | | **17.1%** | **17.3%** | **17.9%** |

**Fig. 13.** Comparison of EASY and FCFS in a 1024 cores system with linear think time.

| EASY/ FCFS | 128 | $Q_{2.5\%}$ | Median | $Q_{97.5\%}$ |
|---|---|---|---|---|
| AWRT | F.lin | 25875.20 | 29462.72 | 36106.23 |
| | E.lin | 23862.72 | 27720.17 | 31576.47 |
| | | *7.8%* | *5.9%* | 12.5% |
| ART | F.lin | 14939.06 | 17321.88 | 20617.80 |
| | E.lin | 16781.50 | 18964.58 | 22017.13 |
| | | **-12.3%** | **-9.5%** | **-6.8%** |
| AWT | F.lin | 13303.00 | 15447.49 | 18586.51 |
| | E.lin | 15055.25 | 17086.67 | 19782.00 |
| | | **-13.2%** | **-10.6%** | **-6.4%** |
| ASD | F.lin | 110.62 | 176.66 | 306.30 |
| | E.lin | 209.74 | 366.45 | 581.06 |
| | | **-89.6%** | **-107.4%** | **-89.7%** |
| WL | F.lin | 237278984 | 259226294 | 285866200 |
| | E.lin | 214225436 | 239972934 | 267924464 |
| | | **9.7%** | **7.4%** | **6.3%** |

**Fig. 14.** Comparison of EASY and FCFS in a 128 cores system with linear think time.

Further analyses of feedback effects can also lead to better scheduler designs. Once we better understand the effects, their causes, and their influence on schedulers, we can exploit them to find better schedules aiming to improve user satisfaction.

## References


1. teikoku Grid Scheduling Framework. http://forge.it.irf.tu-dortmund.de/projects/teikoku/, 2012. online, accessed 11/12/2014.
2. S. Di, D. Kondo, and W. Cirne. Host Load Prediction in a Google Compute Cloud with a Bayesian Model. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 1–11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.
3. D. G. Feitelson. Metrics for Parallel Job Scheduling and Their Convergence. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pages 188–205. Springer Berlin Heidelberg, 2001.
4. D. G. Feitelson. Looking at data. In *IPDPS'08*, pages 1–9, 2008.
5. D. G. Feitelson. Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/, 2008. online, accessed 11/12/2014.
6. D. G. Feitelson. Workload Modeling for Computer Systems Performance Evaluation. http://www.cs.huji.ac.il/∼feit/wlmod/wlmod.pdf, 2014. online, accessed 11/12/2014.
7. C. B. Lee and A. Snavely. On the User-Scheduler Dialogue: Studies of User-Provided Runtime Estimates and Utility Functions. In *International Journal of High Performance Computing Applications*, volume 20, pages 495–506. 2006.
8. D. Lifka. The ANL/IBM SP scheduling system. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pages 295–303. Springer Berlin Heidelberg, 1995.
9. A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das. Towards characterizing cloud backend workloads: Insights from google compute clusters. *SIGMETRICS Perform. Eval. Rev.*, 37(4):34–41, Mar. 2010.
10. U. Schwiegelshohn. How to Design a Job Scheduling Algorithm. In *18th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2014.
11. E. Shmueli and D. G. Feitelson. On simulation and design of parallel-systems schedulers: Are we doing the right thing? *IEEE Transactions on Parallel and Distributed Systems*, 20(7):983–996, 2009.
12. W. Tang, N. Desai, D. Buettner, and Z. Lan. Analyzing and adjusting user runtime estimates to improve job scheduling on the blue gene/p. In *IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1–11, April 2010.
13. N. Zakay and D. G. Feitelson. On identifying user session boundaries in parallel workload logs. In *16th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, pages 216–234, 2012.
14. N. Zakay and D. G. Feitelson. Workload resampling for performance evaluation of parallel job schedulers. *Concurrency and Computation: Practice and Experience*, 26(12):2079–2105, 2014.