Measuring Fragmentation of Two-Dimensional Resources Applied to Advance Reservation Grid Scheduling

Julius Gehr, Jörg Schneider
Technische Universitaet Berlin
{jules,komm}@cs.tu-berlin.de

Abstract

Whenever a resource allocation fails although enough free capacity being available, fragmentation is easily spotted as cause. But how the fragmentation in a system requiring continuous allocations like time schedules or memory can be quantified is hardly analyzed. A Grid environment using advance reservation even combines two-dimensions: time and resource dimension. In this paper a new way to measure the fragmentation of a system in one dimension is proposed. This measure is then extended to incorporate also the second dimension. Extensive simulations showed that the proposed fragmentation measure is a good indicator of the state of the system.

1 Introduction

Fragmentation is a well known effect in resource allocation. A request will be rejected due to fragmentation, if it cannot be allocated to a continuous range of resources while the overall remaining capacity being sufficient to handle it. Therefore, it is rather easy to identify fragmentation as an individual reason for the rejection of a single allocation. On the other hand, describing the general state of an allocation system by quantifying fragmentation is a complex task. Nonetheless, such information could help to compare the effects of a scheduling decision. The fragmentation measure could be used as a forecast how likely future allocations may succeed.

Advance reservation of resources has been identified as a basic requirement to guaranty quality of service in distributed resource management systems like the Grid [1]. Especially, complex Grid jobs composed of interdependent sub-jobs – so-called Grid workflows – require the Grid resource management system to negotiate all start times for all sub-jobs to guarantee the successful execution of the whole job until the given deadline.

Using advance reservation for multi-unit resources, e.g., parallel computer with a number of processors or storage systems introduces a new dimension for fragmentation. It is no longer sufficient to have a continuous time span of a free resource, but there must also be a sufficient amount of free units on the same resource during the whole time span.

In this work we propose a fragmentation measure for resource schedules, i.e., the fragmentation of the time dimension. This measure is further extended to model resource allocation schedules of multi-unit resources. The fragmentation measure will be derived from former fragmentation measures and further discussed using exemplary schedules. To evaluate the resulting fragmentation measure, the correlation between the measured fragmentation of a schedule and the future rejection rate will be analyzed. In the next sections the specific application domain Grid workflow scheduling is presented and the various reasons for rejecting an allocation request in this setting are discussed.

2 Two-dimensional Resource Allocation in the Grid

Nowadays, most scientists have already access to multiple high performance computers. Still it is a rather inconvenient task to select the right one to execute a specific task. The idea of Grid computing is to provide a single entry point (Grid broker) for all these resources, which according to some allocation strategy takes care of selecting the resource, reserving it and monitoring the execution. By inter-organizational cooperation, such a Grid can grow further and thereby incorporating additionally resource types as network, storage or even scientific instruments.

As the Grid couples pre-managed and rather big resources, the user typically doesn't book the whole resource but a small partition. Usually due to the distributed nature of the Grid, this partition has to be within the same physical resource. Therefore, the Grid broker has to find for each allocation a single resource with sufficient free resources.

Grid workflows (see Figure 1) are complex Grid jobs consisting of a number of inter-dependent tasks. In order to guarantee the successful execution in the specified order,

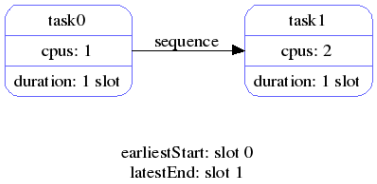


Figure 1: Simple workflow example with sequence dependency.

the Grid broker negotiates fixed start times for each sub-job with all booked resources. Using so-called advance reservations therefore increases the search space for a set of valid allocations. One dimension is the start time for each job and the other is the resource having sufficient free capacity.

Scheduling Grid workflows has a high complexity. Therefor, advance reservation Grid broker use often a heuristic that may reject a workflow even if there is a valid constellation for allocating the Grid workflow. However, a workflow may only accepted if a valid constellation has been definitely found. In the next section we further discuss the various reasons for a request rejection.

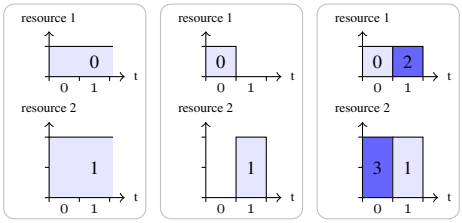
3 Fragmentation - Reason for Rejected Allocation Requests

A complex Grid job like a Grid workflow can be rejected by a number of reasons. In this section we discuss all possible reasons for rejection and how they could be improved or at least detected beforehand.

Assuming a new Grid workflow *wf1* as depicted in Figure 1 is submitted to the Grid broker managing a small Grid with two compute resources with 1 respectively 2 CPUs. The Figures 2 and 3 show for each of the following identified reasons exemplary schedule-situations.

1. **Rejection because of premature abort of the scheduling unit.** The simplest cause of rejection is that the scheduling unit was not able to find a valid assignment even though one existed. As discussed before, this may happen due to strict run-time requirements of the heuristic scheduler. In this case, it is the fault of the scheduler and it could be solved by improving the scheduling algorithm or allowing for longer processing time.

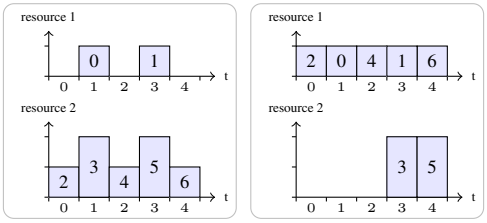
2. **Rejection because of high utilization.** If there is no vacancy on the resources as in Figure 2a the scheduling unit will reject the request. There is nothing one could do about it and the resource owners are interested in high utilization anyway since it maximizes their earnings.



(a) Fully utilized resources.

(b) *wf1* scheduled for the first time.

(c) *wf1* and *wf1'* can be scheduled together.



(d) Fragmented free blocks.

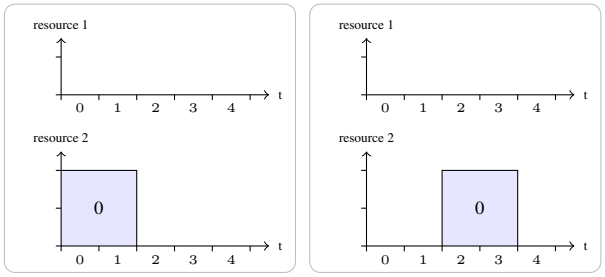
(e) Large free block.

Figure 2: Causes of Rejection

3. **Rejection because of workflow structure.** In Figure 2b *wf1* was successfully scheduled for the first time, but it is not possible to assign this workflow a second time onto the same resources. However, a modified version *wf1'* which is *wf1* but with a reversed sequence dependency could be assigned as shown in Figure 2c. Therefore, the inability of assigning *wf1* twice does not result out of utilization since *wf1'* demands the same quantity of resources as *wf1*. However, rejection because of the inner structure of workflows cannot be measured in a way that a certain schedule-situation is more or less prone to structural problems. Therefore, this problem cannot be identified beforehand.

4. **Rejection because of fragmentation.** If free parts of the resources are shattered in space and time rejection because of fragmentation will appear. In the situation depicted in Figure 2d, it is evident that *wf1* cannot be scheduled. However, if optimization could compact the reservations in order to form a large free block as in Figure 2e, *wf1* will then be admitted. Since Figure 2d and Figure 2e have the same utilization it becomes clear that fragmentation is a cause of rejection in its own right.

5. **Rejection because of unfavorable previous decisions** For an example on this cause a simple workflow consisting of only one sub-job is considered. If this workflow was assigned as in Figure 3a, the rejection of *wf1* neither comes from fragmentation nor from



(a) Blocked first slots. (b) Unblocked first slots.

Figure 3: Rejection because of previous decision.

utilization since the utilization and fragmentation are both very low. The assignment made in Figure 3a turns out to be unfavorable, the workflow *wf1* could have been admitted if the previous assignment had been as in Figure 3b.

This example may suggest that it is better to schedule all workflows as late as possible. This however does not solve the problem, since *wf1* may also have demanded slot 1 and 2. The problem rather origins from the inability to foresee the future. Usually, the scheduler considers only the requested Grid workflow. So by changing the reservations of already admitted workflows, the problem could be partially solved at the expense of longer run times.

Since all other causes of rejection can be handled or are inevitable without changing the Grid setup, we further analyze the fragmentation as a way to describe the state of the Grid.

4 Related Work

Using advance reservation is an important allocation strategy, widely used, e.g., in Grid tool kits such as Globus [3] or VRM [1], as they provide simple means for planning of resources and in particular co-allocations of different resources. Besides flexibility and easy support for co-allocations, e.g., while processing complex workflows, advance reservations also have other advantages such as an increased admission probability when reserving sufficiently early, and reliable planning for users and operators. In contrast to the synchronous usage of several different resources, where also queueing approaches are conceivable, advance reservations have a particular advantage when time-dependent co-allocation is necessary. Support for advance reservations has been integrated into several management systems for distributed and parallel computing [10]. In [1], advance reservations have been identified as essential for a number of higher level services, such as SLAs.

It remains to be examined how fragmentation can be measured in the special domain of grid resources. The nearest solution would be to reuse ideas of how to measure fragmentation in other domains, e.g., in file systems and fragmentation in main memory.

A detailed performance comparison of the 4.4BSD Log-structured File System and the 4.4BSD Fast File System (FFS) is presented in [9]. The following three factors were found to be relevant for contributing to free space fragmentation in a FFS: high file turnover, high utilization and file system age. This was further analyzed by comparing FFS file systems of file servers under real workloads with the performance of empty FFS file systems over a period of nine months. It was implied that the observed performance degradation is due to the fragmentation of the file system. However, the actual fragmentation of the free space blocks in the file systems as well as the non-contiguity of files were not quantified.

The file system approaches OBFS, XFS and yFS were presented in [12, 11, 14]. Architectures and details regarding the implementation e.g.: usage of logs or B* or B+ trees were discussed and performance studies were done, however without actually quantifying fragmentation.

Sears et.al. [8] dealt with the problem of whether a binary large object (BLOB) should be stored in the file system or in a database. The size of a BLOB was considered as a main factor concerning this question and it was studied how the performance of read and write operations in file systems and databases depends on the size of the BLOB. Furthermore, performance degradation due to fragmentation was examined by using fragments per object as a measure. This approach will be discussed later.

In [4, 13] the characteristics of dynamic memory allocators were studied. The studied allocators manage the general purpose heap, which can be requested or freed by a program at any time. They have to deal with problems such as finding a free block for satisfying a `malloc()` request, choosing one block out of many possible ones, splitting a block which is larger than the requested one, coalescing two or more adjacent freed blocks, demanding more memory from the operating system with e.g. `sbrk()` to serve a `malloc()` request. It is outlined that previous studies wrongfully use synthetic traces and therefore disregard important regularities in real program's memory usage patterns. This study uses a lot of common programs gcc, perl, etc. and two values were measured over the time: First, live memory as requested by the program. Second, the memory used by the allocator to fulfill the requests. Four methods of how to measure fragmentation were introduced, e.g.: The amount of memory used by the allocator divided by the amount of memory requested by the program, averaged

over all points in time. It was stated that the question of whether a high degree of fragmentation is present at a certain situation or not is dependent on the sizes of the following `malloc()`s. This is due to the fact that a program which requests only very small portions of memory can be satisfied with small splintered pieces of memory and therefore the fragmentation has no impact. Although being appropriate for this application, these kind of measures are rather measuring the wastage of memory which indirectly also serves as a measure of fragmentation. Furthermore, the domain of memory management does not map onto the domain of grid resources very well because there is no match for `sbrk()` (which increments the data segment size) within the grid domain. In addition the main memory can be considered as homogeneous which is not true for grid resources. For the main memory, apart from effects of locality, it does not matter whether object1 is in cell1 and object2 is in cell2 or vice versa. However, for grid resources it does matter whether reservation1 is in slot1 and reservation2 is in slot2 since slot2 could be too late for reservation1. Analogous, it is true for the other dimension that it does matter whether reservation1 is assigned to resource1 and reservation2 is assigned to resource2 since resource2 may not be capable of handling reservation1.

Another measure of fragmentation was suggested in [7] which is also part of the memory allocator domain: $F = \frac{MaxHeapSize - MaxLiveBytes}{MaxLiveBytes}$. As in the case before concerning grid resources, there exists no match for the heap as well. The proposed measure is the memory wastage. It is not appropriate for the geometric aspect of how much the free blocks have been splintered.

Finally, in [5] it was proposed to measure fragmentation as $F = 1 - \frac{LargestFreeBlock}{AllFreeMemory}$ which will be discussed later on.

5 Measuring Fragmentation

In the last section, approaches already existing for measuring fragmentation have been studied and although being mostly not applicable in the domain of grid resources, the interesting ones will be further discussed. Later on, a new measurement of fragmentation will be proposed. A drawback which is inherent to all approaches studied so far is their limitation to one dimension. However, grid resources have two dimensions: time and resource capacity, e.g., number of CPUs or bandwidth. A procedure which transfers these one-dimensional approaches into two-dimensional ones will be introduced.

Table 1 gives an overview on the symbols used in the following section.

Symbol	Explanation
b_i	width of the i-th booked-block
f_i	width of the i-th free-block
n	number of free-blocks
m	number of booked-blocks
f	$f = \{f_i \mid 1 \leq i \leq n\}$
$\overline{f}$	f's average
F	fragmentation with $F \in [0, 1]$ and 0 for the lowest fragmentation
U	utilization generated by a schedule
N	non-linear utilization
R	rating, used to compare different schedules
p	controls how fast the fragmentation converges to 1

Table 1: Used symbols

5.1 Fragmentation in a Single Dimension

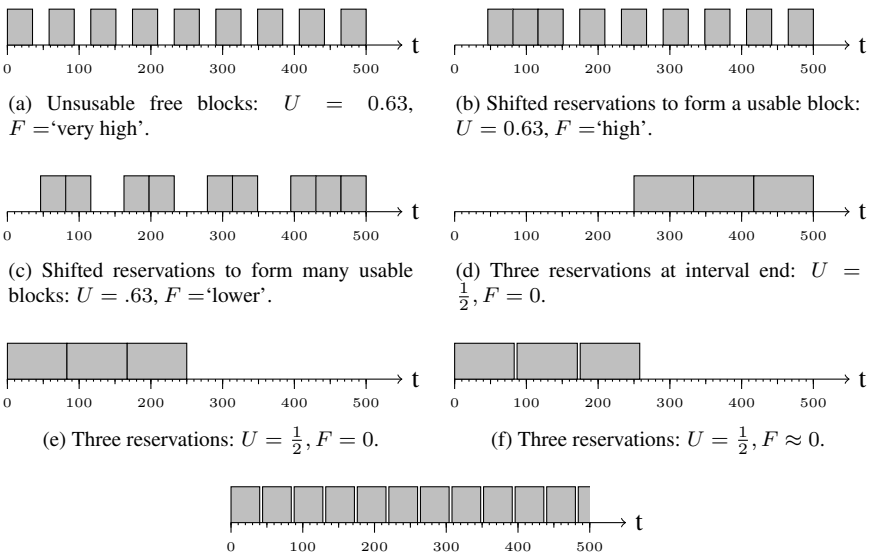
In order to determine a measure of fragmentation, examples with distinct schedule-situations were used. A small subset of these examples is depicted in Figure 4. These situations were rated by using visual judgment. Situations with one large contiguous chunk of free space and very few and very small additional splinters were regarded as situations with nearly zero fragmentation. The estimated fragmentation got as worse as the free space got splintered. In the following, a number of formulas were examined whether they fulfill these requirements.

The most simple approach is the one from [8] which suggests to measure fragmentation as: $F = 1 - \frac{1}{n}$, with n being the number of free-blocks. A situation as in Figure 4f will be rated with $F = \frac{2}{3}$ whereas it measures $F = 0$ for Figure 4e. Both situations are nearly identical but rated substantially different by the proposal. The problem of this measure is that the sizes of the free blocks are not taken into account. This approach is not suitable because it is reasonable to neglect small free blocks.

In contrast to the last approach, in Jan Lindblad's proposal for measuring fragmentation [5] the size of the largest free block is considered:

$$F = 1 - \frac{max\{f\}}{\sum_{i=1}^{n} f_i}$$

The sizes of the other blocks are however ignored and it rates different fragmentation situations as equal, e.g.: in Figure 4b the free space without the largest block is shattered into many small and unusable splinters. Whereas, if the free space is divided in blocks of the same size as the largest free block (as in Figure 4c) the resource would be



(a) Unsusable free blocks: $U = 0.63$, $F =$'very high'.

(b) Shifted reservations to form a usable block: $U = 0.63$, $F =$'high'.

(c) Shifted reservations to form many usable blocks: $U = .63$, $F =$'lower'.

(d) Three reservations at interval end: $U = \frac{1}{2}, F = 0$.

(e) Three reservations: $U = \frac{1}{2}, F = 0$.

(f) Three reservations: $U = \frac{1}{2}, F \approx 0$.

(g) High utilization, very very high fragmentation.

Figure 4: Reserved Timeslots for a single Resource

much more utilizable. However this approach rates both situations with the same fragmentation $F = 0.25$. This approach is also inappropriate.

In Figure 4b and in Figure 4a the fragmentation seems grave and the average free block size is small. In contrast to Figure 4c and Figure 4e where the fragmentation seems to be better and the average free block size is larger. Therefore, the average free block size can be used to calculate fragmentation as:

$$F = 1 - \frac{\overline{f}}{\sum_{i=1}^{n} f_i}$$

This method works perfectly for Figure 4e and yields 0. Figure 4f containing free blocks 4, 4, 242 is rated as $\frac{2}{3}$. However, this approach is mathematically identical to the first one since the arithmetic mean is $\overline{f} = \frac{1}{n}\sum_{i=1}^{n} f_i$.

The arithmetic mean may not be appropriate anyhow, since it is very fragile to outliers. Therefore, it is reasonable to consider other averages not so sensitive to outliers. Another average, well known for its immunity to outliers, is the median which can be found by sorting the values and choosing the one in the middle. In Figure 4f the observed values are $(4, 4, 250)$ and therefore the median is 4. The median has identified the 250 as being an outlier which might be reasonable in some cases but not in this one. Actually, both values of 4 are outliers whereas the large value is not. In order to boost the influence of the large values, they can be raised to the power of a number greater than 1.

This leads to the new formula to measure fragmentation:

$$F = 1 - \frac{\sum_{i=1}^{n} f_i^p}{\left(\sum_{i=1}^{n} f_i\right)^p}$$

Several consequences arise for this kind of fragmentation measurement. First, it holds $\sum_{i=1}^{n} f_i^p \leq (\sum_{i=1}^{n} f_i)^p$. From this it conveniently follows that $F \in [0, 1)$.

Furthermore, two additional statements can be made about this way of fragmentation measuring. First, $F = 0$ if $n = 1$. Second, $F \to 1$ if $n \to \infty$ and blocks being sized equally. In the case of $f_1 = f_2 = \ldots = f_n$ the fragmentation formula can be simplified.

$$\begin{aligned} F &= 1 - \frac{\sum_{i=1}^{n} f_i^p}{\left(\sum_{i=1}^{n} f_i\right)^p} = 1 - \frac{\sum_{i=1}^{n} f_1^p}{\left(\sum_{i=1}^{n} f_1\right)^p} \\ &= 1 - \frac{n f_1^p}{(n f_1)^p} = 1 - \frac{n f_1^p}{n^p f_1^p} \\ &= 1 - \frac{1}{n^{p-1}} \end{aligned}$$

It converges as fast as large p gets. For $p = 2$ and equally sized fragments the measure transforms to $F = 1 - \frac{1}{n}$ which is the approach from [8]. E.g., for $n = 1, 2, 3, \ldots, 100$ fragmentation becomes $F = 0, \frac{1}{2}, \frac{2}{3}, \ldots, \frac{99}{100}$. This converging

Listing 1: 2D fragmentation measure

```
float calc_2D_fragmentation() {
  float frag = 0.0;
  int num = 0;
  float new_frag = 0.0;

  for(int demand = 1 to MAX_DEMAND) {
    new_frag = calc_1D_fragmentation(
      tag_occupied_vacant(demand)
    );

    if(new_frag != 0) {
      frag += new_frag;
      ++num;
    }
  }

  return frag / num;
}
```

behavior for growing values of n is essential for fragmentation measures and naturally it is also fulfilled by $F = 1 - \frac{1}{n}$.

However, the insensitivity to small outliers has to be tested additionally. Using the example from Figure 4f, for $p = 2$ the fragmentation equals:

$$F = 1 - \frac{4^2 + 4^2 + 250^2}{(4 + 4 + 250)^2} = 1 - \frac{62532}{66564} \approx 0.06$$

which is obviously better than $\frac{2}{3}$ yielded by $F = 1 - \frac{1}{n}$. This approach has the necessary property of being resistant to small negligible fragments as long as one large fragment exists.

As in the discussion before, in the following analysis $p = 2$ will be used.

5.2 Adding the Second Dimension

Even though a promising candidate has been found, it still lacks the ability to measure fragmentation in two dimensions, i.e., for a time schedule of multi-unit resources. It regards time slots as occupied or vacant but it does not deal with the amount of free resources (e.g. 1 cpu or 10 cpus) within the slots. However, this binary assumption matches the point of view of the scheduler while scheduling a new task. All time slots become either occupied or vacant for that specific task according to its demands on the resource. Thus, for a specific demand of resources the 1D measure can be applied. And in order get a 2D measure the average over all possible demands of this resource can be calculated. In Listing 1 such a calculation is shown. This approach can be further improved by using the relative frequency of occurrences for every demand as average weights.

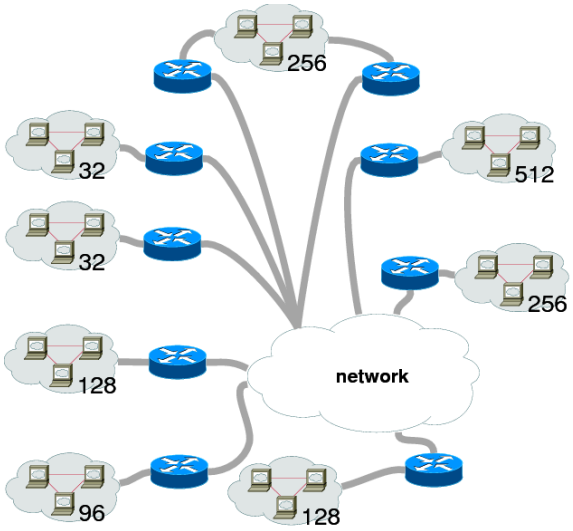


Figure 5: The simulated Grid with 8 compute resources.

6 Evaluation

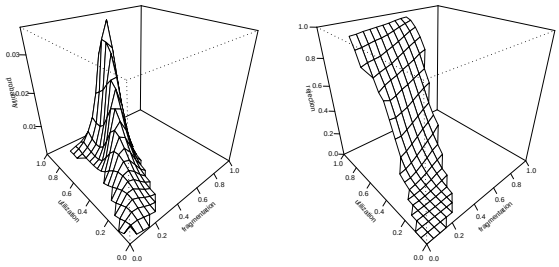
As discussed in Section 3, the rejection ratio should be highly influenced by the measured fragmentation. To evaluate the proposed fragmentation measure the rejection rate was compared between situations where the measured fragmentation differed, while all other parameters have been constant.

Because, the actual distribution of the job sizes, the job durations etc. does not impact the general quality of the results, even when using simple models [6], the simulations were made using a synthetic workflow model.

The modeled Grid (see Figure 5) consisted of eight compute resources with 512, 256, 256, 128, 128, 96, 32, and 32 CPUs. As only the fragmentation in the compute resources was analyzed, all computers were connected to a network without any bandwidth or latency restrictions. The simulation was caried out with the simulation mode of the VRM Grid broker [1] using the Grid workflow scheduler described in [2].

6.1 Workload Construction

At first, more than 70,000 independent workloads have been created, each of them containing 10,000 workflows. Each workflow is assumed to be reserved in advance with an interarrival time distributed normally with parameters $\mu = 100$ and $\sigma = 1000$. The decision against the exponential distribution was made because the exponential distribution lacks the ability of separately choosing variance and mean. In this simulation setup, there is no way of controlling the simulations in such a way that specific states of utilization and fragmentation can be achieved. However, a



(a) Spatial probability density in dependence from fragmentation and utilization.

(b) Rejection rate in dependence from fragmentation and utilization.

Figure 6: Impact of fragmentation and utilization.

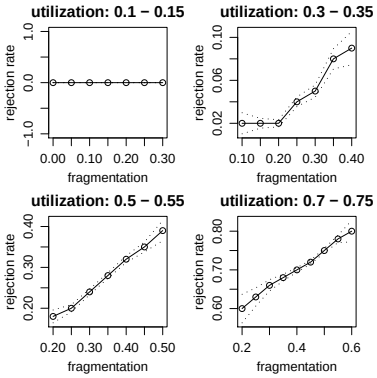


Figure 7: Impact of fragmentation on the rejection rate.

high standard deviation had the consequence that the simulations covered a sufficiently large area in the utilization-fragmentation-domain. The book-ahead time was normally distributed, too, with parameters $\mu = 150$ and $\sigma = 300$. Each workflow consists of a number of tasks which are uniformly distributed over $\{4, 5, \ldots, 9, 10\}$. Task duration and requested processors were also uniformly distributed over the interval $[250, 750]$ and $\{2, 4, 8, 16, \ldots, 256\}$. The deadline of the workflow is the sum of all task durations and a normally distributed random value with $\mu = 0$ and $\sigma = 300$. The creation unit of the synthetic workflows tries to associate each task with zero to three dependencies uniformly. These dependencies are composed of 66% sequential dependencies and 33% synchronous dependencies. Creating a new dependency fails if it creates a loop within the workflow. At most, 20 assignment trials per dependency are done.

6.2 Workload Processing

Each workload was processed by the simulated grid and it has been traced if a distinct workflow could successfully be assigned to the grid. Before beginning a scheduling process, utilization and fragmentation were also recorded. Each simulation run has produced a list of $(rejection, U, F)$ samples with $rejection \in \{0, 1\}$ and $U \in [0, 1]$ and $F \in [0, 1)$. Unfortunately, these triplets are not independent, e.g., if a workflow has been rejected the successor is likely to be rejected, too. The closer their arrival times are the more likely is the rejection of the successor since the probability is high that the reason remains which caused the first rejection. Furthermore, the farther the arrival times are distributed over time, the smaller is the autocorrelation of the samples. In order to collect independent data and to keep mathematics simple, most samples were discarded. For each independent run, the first hundred samples were discarded which skips the initial transient phase. After each used sample, a number of samples were discarded in order to get independent data. These numbers were chosen randomly out of $\{100, 101, \ldots, 200\}$. Random values were used to circumvent possible existing periodic behavior.

The samples were divided into different categories. An distinct sample $(rejection, U, F)$ belongs to a distinct category $S_{u,f}$ with $u, f \in \{0,\ 0.05,\ 0.1,\ \ldots,\ 0.9,\ 0.95\}$ if and only if $u \leq U < u + 0.05\ \wedge\ f \leq F < f + 0.05$. The estimated rejection rate for a distinct category $S_{u,f}$ equals:

$$r_{u,f} = \frac{\sum_{i \in S_{u,f}} i.rejection}{|S_{u,f}|}$$

6.3 Simulation Results

After extensive simulations have been done, the impact of the new fragmentation measure on the rejection rate was examined.

First, it has to be mentioned that the system does not cover all possible $S_{f,u}$ categories. In Figure 6a, the state probability of each category is depicted. It can be observed that the utilization covers nearly the whole domain from 0% to 90% while the fragmentation only covers the area from 0% only to 60%. Due to the slotted time, free blocks have at least the size of a slot and the number of free slots within an interval is bounded. Hence, the fragmentation is bounded, too. Considering the slot size the search interval size and the minimal task size, the boundary caused by the slotted time is approximately 95%. Thus, there must be another reason for the fragmentation being bounded. The used scheduler[2] tends to schedule all tasks as early as possible. This approach seems to avoid severe fragmentation. Further investigations is planned as future work.

In Figure 6b, the rejection rate is depicted. It can be clearly observed that the rejection rate grows with higher utilization or with higher fragmentation. In order to clarify

the impact of fragmentation on the rejection rate, in Figure 7 the rejection rate is plotted in dependency on fragmentation for some fixed values of utilization. A confidence interval for the rejection rate is depicted as dotted line for a confidence of 99.993%. It can be clearly observed that for very low utilization, fragmentation remained quite low and no rejection appeared. For more realistic scenarios, the utilization is between medium and high. In the case of 50% utilization, the observed fragmentation is between 20% and 50%. The rejection rate is between 18% and 40% yielding a difference of more then 20%. Thus the fragmentation as measured by the new proposed measure has a huge impact on the rejection rate.

7 Conclusion and Outlook

In this paper a new fragmentation measure for judging the state of one-dimensional allocation systems like time schedules was proposed. This measure was further extended to also cope with multi-unit resources over time, thus handling both dimensions within a single measure. Extensive simulations showed, that the proposed fragmentation measure reflects the state of the allocation system very well.

Future work will deal with further analyzing the behavior of the new fragmentation measure, i.e., with other values of p or in the context of other two-dimensional resource allocation systems. The next step would be to enhance the scheduler to avoid allocations leading to a high fragmentation.

References

[1] L.-O. Burchard, M. Hovestadt, O. Kao, A. Keller, and B. Linnert. The virtual resource manager: An architecture for SLA-aware resource management. In *4th Intl. IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGrid) 2004, Chicago, USA*, 2004.

[2] J. Decker and J. Schneider. Heuristic scheduling of grid workflows supporting co-allocation and advance reservation. In B. Schulz, R. Buyya, P. Navaux, W. Cirne, and V. Rebello, editors, *7th Intl. IEEE Intl. Symposium on Cluster Computing and the Grid (CCGrid07)*, pages 335–342, Rio de Janeiro, Brazil, May 2007. IEEE CS Press.

[3] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *7th International Workshop on Quality of Service (IWQoS), London, UK*, pages 27–36, 1999.

[4] M. S. Johnstone and P. R. Wilson. The memory fragmentation problem: Solved? *SPNOTICES: ACM SIGPLAN Notices*, 34, 1999.

[5] J. Lindblad. Handling memory fragmentation. `http://www.ednasia.com/article.asp?id=182` (visited December 1th 2006).

[6] V. Lo, J. Mache, and K. Windisch. A Comparative Study of Real Workload Traces and Synthetic Workload Models for Parallel Job Scheduling. In *4th Workshop on Job Scheduling Strategies for Parallel Processing, Orlando, USA*, volume 1459 of *Lecture Notes in Computer Science (LNCS)*, pages 25–46. Springer, 1998.

[7] M. S. Neely. An analysis of the effects of memory allocation policy on storage fragmentation. Master's thesis, University of Texas at Austin, 1996.

[8] R. Sears, C. V. Ingen, and J. Gray. To BLOB or not to BLOB: Large object storage in a database or a filesystem? Technical Report MSR-TR-2006-45, Microsoft Research (MSR), Apr. 2006.

[9] M. Seltzer, K. A. Smith, H. Balakrishnan, J. Chang, S. McMains, and V. Padmanabhan. File system logging versus clustering: A performance comparison. In *Proceedings of the USENIX 1995 Technical Conference*, pages 249–264, New Orleans, LA, USA, Jan. 16–20 1995.

[10] D. Snell, M. Clement, D. Jackson, and C. Gregory. The Performance Impact of Advance Reservation Meta-scheduling. In *6th Workshop on Job Scheduling Strategies for Parallel Processing, Cancun, Mexiko*, volume 1911 of *Lecture Notes in Computer Science (LNCS)*, pages 137–153. Springer, 2000.

[11] A. Sweeney. Scalability in the XFS file system. In *USENIX Annual Technical Conference*, pages 1–14, 1996.

[12] F. Wang, S. A. Br, E. L. Miller, and D. D. E. Long. OBFS: A file system for object-based storage devices, May 04 2004.

[13] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles. Dynamic storage allocation: A survey and critical review. *Lecture Notes in Computer Science*, 986:1–??, 1995.

[14] Z. Zhang and K. Ghose. yFS: A journaling file system design for handling large data sets with reduced seeking. In *FAST*. USENIX, 2003.