Dipartimento di Informatica
Università del Piemonte Orientale "A. Avogadro"
Via Bellini 25/G, 15100 Alessandria
http://www.di.unipmn.it

Knowledge-Free Scheduling Algorithms for Multiple Bag-of-Task Applications on Desktop Grids

Authors: Cosimo Anglano, Massimo Canonico
{*cosimo.anglano,massimo.canonico*}*@di.unipmn.it*

TECHNICAL REPORT TR-INF-2009-02-01-UNIPMN
(February 2009)

The University of Piemonte Orientale Department of Computer Science Research Technical Reports are available via WWW at URL `http://www.di.mfn.unipmn.it/`.
Plain-text abstracts organized by year are available in the directory

Recent Titles from the TR-INF-UNIPMN Technical Report Series

2008-09 *Case-based management of exceptions to business processes: an approach exploiting prototypes*, Montani, S., December 2008.
2008-08 *The ShareGrid Portal: an easy way to submit jobs on computational Grids*, Anglano, C., Canonico, M., Guazzone, M., October 2008.
2008-07 *BuzzChecker: Exploiting the Web to Better Understand Society*, Furini, M., Montangero, S., July 2008.
2008-06 *Low-Memory Adaptive Prefix Coding*, Gagie, T., Nekrich, Y., July 2008.
2008-05 *Non deterministic Repairable Fault Trees for computing optimal repair strategy*, Beccuti, M., Codetta-Raiteri, D., Franceschinis, G., July 2008.
2008-04 *Reliability and QoS Analysis of the Italian GARR network*, Bobbio, A., Terruggia, R., June 2008.
2008-03 *Mean Field Methods in performance analysis*, Gribaudo, M., Telek, M., Bobbio, A., March 2008.
2008-02 *Move-to-Front, Distance Coding, and Inversion Frequencies Revisited*, Gagie, T., Manzini, G., March 2008.
2008-01 *Space-Conscious Data Indexing and Compression in a Streaming Model*, Ferragina, P., Gagie, T., Manzini, G., February 2008.
2007-05 *Scheduling Algorithms for Multiple Bag-of-Task Applications on Desktop Grids: a Knowledge-Free Approach*, Canonico, M., Anglano, C., December 2007.
2007-04 *Verifying the Conformance of Agents with Multiparty Protocols*, Giordano, L., Martelli, A., November 2007.
2007-03 *A fuzzy approach to similarity in Case-Based Reasoning suitable to SQL implementation*, Portinale, L., Montani, S., October 2007.
2007-02 *Space-conscious compression*, Gagie, T., Manzini, G., June 2007.
2007-01 *Markov Decision Petri Net and Markov Decision Well-formed Net Formalisms*, Beccuti, M., Franceschinis, G., Haddad, S., February 2007.
2006-03 *New challenges in network reliability analysis*, Bobbio, A., Ferraris, C., Terruggia, R., November 2006.
2006-03 *The Engineering of a Compression Boosting Library: Theory vs Practice in BWT compression*, Ferragina, P., Giancarlo, R., Manzini, G., June 2006.
2006-02 *A Case-Based Architecture for Temporal Abstraction Configuration and Processing*, Portinale, L., Montani, S., Bottrighi, A., Leonardi, G., Juarez, J., May 2006.
2006-01 *The Draw-Net Modeling System: a framework for the design and the solution of single-formalism and multi-formalism models*, Gribaudo, M., Codetta-Raiteri, D., Franceschinis, G., January 2006.
2005-06 *Compressing and Searching XML Data Via Two Zips*, Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S., December 2005.

Knowledge-Free Scheduling Algorithms for Multiple Bag-of-Task Applications on Desktop Grids

Cosimo Anglano, Massimo Canonico
C. Anglano and M. Canonico are with the Università del Piemonte Orientale, Dipartimento di Informatica, Alessandria (Italy).
E-mail:cosimo.anglano@unipmn.it, massimo.canonico@unipmn.it

Abstract

Desktop Grids have the potential to provide an effective, low-cost solution to the computing needs of a variety of distributed applications consisting in a set of independent tasks. However, such a potential can be exploited in practice only if suitable scheduling strategies are employed. In this paper we propose a set of *knowledge-free* scheduling algorithms (that is, they do not require any information concerning either resource status or application characteristics) that are able to effectively schedule a set of competing applications simultaneously submitted to a Desktop Grid, unlike previous solutions that are able to deal only with a single application at a time. We study, via simulation, the performance of these strategies, as well as their ability of efficiently using the available resources, for a wide range of Desktop Grid configurations (ranging from Volunteer Computing systems to Enterprise Desktop Grids) and application workloads. Our results show that the scheduling algorithms we propose outperform naive alternative like FCFS (the *de facto* standard for Desktop Grids) both in terms of application performance and efficient resource usage. Furthermore, these results enable us to identify which one of the proposed algorithms should be used for a given combination of Desktop Grid configuration and application workload.

I. Introduction

The widespread diffusion of the Internet has created a new much large scale opportunity for Grid computing. Millions of desktop PCs, whose computing and storage capacity is used only in a small part, are indeed connected to wide-area networks both in the enterprise and in the home. The aggregation of these independent resources into integrated computing platforms, achieved by means of suitable middlewares (e.g., *OurGrid* [1], *BOINC* [2], *XtremWeb* [3], *United Devices* [4], and *Data Synapse* [5]), can provide unprecedented amounts of raw computing power to distributed applications. These platforms, known in the literature as *Desktop Grids* [6], [7], have been profitably used in a large number of projects (e.g., *The Great Internet Mersenne Prime Search* [8], *Seti@home* [9], *Folding@home* [10], *FightAids@home* [11], just to name a few), thanks to their ability of providing amounts of raw computing power that far exceed the capabilities of traditional Grid platforms at a fraction of their costs. The Desktop Grid paradigm, born in academic settings, has been quickly adopted in the enterprise where, however, it has been interpreted in a different way. As a matter of fact, academic Desktop Grids typically (but not only) include machines owned by independent users that voluntarily "donate" to the system a fraction of capacity of their machines (hence their name *Volunteer Computing Systems*). Conversely, Desktop Grids used within a single enterprise (denoted as *Enterprise Desktop Grids*) comprise resources owned by that enterprise only, that in this way can obtain a degree of data and application confidentiality higher than those attainable by Volunteer Computing systems.

Regardless of their academic or enterprise nature, however, Desktop Grids share the same core features: like traditional Grids, they are characterized by a wide resource distribution and heterogeneity, but unlike them they use resources that are not exclusively dedicated to run Grid applications. As a consequence, these resources exhibit a much higher degree of volatility, since they may be reclaimed by the respective owners at any time without any advance notice, and without caring about the application that was using it.

For this reason, Desktop Grids are generally considered better suited to the execution of loosely-coupled parallel applications, that are able to tolerate the failure of individual application processes much better than tightly-coupled ones. Among these applications, *Bag-of-Tasks* [12], [13] (or simply *BoTs* for brevity) – parallel applications whose tasks are completely independent from one another – have been shown [14] to be particularly suited to Desktop Grids and consequently, despite their simplicity, are used in a variety of domains, such as parameter sweeps, simulations, fractal calculations, computational biology, and computer imaging.

In order to enable BoTs to profitably exploit Desktop Grids, suitable scheduling strategies, able to deal with the heterogeneity of resources, the fluctuations in the performance they deliver because of the simultaneous execution of competing applications, and their failures due to their crashes/reboots or unplanned departures, must be adopted. In response to this need, various scheduling algorithms that attempt to minimize the *makespan* of BoTs (that is, the time taken to execute all the tasks in a bag) in spite of the above problems have been proposed in the literature [14]–[19]. These algorithms, however, are able to deal only with a single BoT at a time so, as shown later in this paper, when the workload consists in a set of BoTs concurrently submitted by a community of potentially distinct users, are not able to properly perform, and yield higher makespans for individual BoTs, lower resource utilization, or both. Consequently, these algorithms are appropriate for situations where there is a single BoT that exclusively uses a Desktop Grid infrastructure (e.g., like in volunteer-computing projects [9], [20]), but they fail to provide an adequate solution when the same infrastructure is shared by many competing applications. Such scenarios, however, are increasingly becoming commonplace both in the academia and in the enterprise. On the one hand, Desktop Grid infrastructures that federate resources belonging to independent (small) research institutions (such as *ShareGrid* [21]) are becoming an established reality, and the workloads they run are naturally composed of applications of different types independently submitted by distinct user communities. On the other hand, similar situations arise also in the enterprise, where different R&D groups share the same Desktop Grid infrastructure to run different applications without coordinating among them to ensure the exclusive usage of the infrastructure. Therefore, scheduling algorithms able to properly schedule a set of BoT applications concurrently submitted for execution to the same Desktop Grid are crucial in order to exploit the potential provided by these platforms.

In spite of the above considerations, the analysis of the relevant literature reveals that the problem of scheduling *multiple* BoTs concurrently submitted for execution to the same Desktop Grid has not been studied yet in a systematic way. This paper fills this gap by proposing a set of *knowledge-free* scheduling algorithms for multiple BoTs (i.e., algorithms not relying on any information concerning the status of resources or applications), and by studying their performance for a large set of Desktop Grid configurations and workloads. Our results clearly show that the scheduling algorithms proposed in this paper are able to effectively schedule application workloads consisting of multiple BoT submitted concurrently, and that they outperform existing scheduling algorithms for Desktop Grids, that – as already discussed – assume that only a single BoT must be scheduled at a given time, that instead result in unacceptably low application performance and resource utilization. Furthermore, we are also able to identify which one of the various algorithms we propose is better suited to a specific combination of Desktop Grid configuration and application workload.

The rest of the paper is organized as follows. Section II discussed related work, while in Section III we precisely define the scheduling problem for multiple BoTs, and we present our scheduling algorithms. In Section IV we describe the results we obtained in our evaluation experiments. Finally, Section V concludes the paper and outlines future research work.

II. Related work

In the recent past, the problem of scheduling *individual* BoTs on Desktop Grid has been actively studied. The scheduling algorithms proposed in the literature can be classified either as *knowledge-based* or *knowledge-free*. Knowledge-based algorithms [15], [17], [18] assume that the scheduler knows and

exploits various amounts of information concerning resources (e.g., the computing power they deliver to applications, their availability, etc.), applications (e.g., the execution times of the tasks). However, it has been observed [14] that in Desktop Grids the information concerning resource status may be very hard to collect (because of resource volatility) and is often inaccurate (because of resource contention). Furthermore, the estimation of task execution times on heterogeneous, shared resources is still an open research problem for which only partial solutions exist [22], [23]. Knowledge-free strategies [14], [16], [19], that instead do not rely on any system or application information, have therefore been proposed as a solution to the above problem, and have been shown [14], [24] to be able to obtain performance comparable to knowledge-based ones at the price of using more resources than strictly necessary. Therefore, in situations where plenty of computing resources are available, they represent a viable solution.

Although the above scheduling algorithms are able to effectively schedule BoTs in the hypothesis that they arrive one at a time (that is, at any single time only one BoT is present in the system), they do not consider scenarios in which *multiple* BoTs must be scheduled. Existing approaches to schedule multiple BoTs use either very simple algorithms (e.g., FCFS [3], [25] or random selection [24]) that – as shown later in this paper – fail to provide adequate performance, or algorithms that adopt a knowledge-based approach and pose restrictions on the communication topology of the Desktop Grid [26].

Unlike the above solutions, the scheduling strategies presented in this paper are knowledge-free and do not pose restrictions on the architecture of the Desktop Grid.

III. Scheduling algorithms for multiple Bag-of-Task applications

In this paper we consider the problem of scheduling a set BoTs concurrently submitted to a Desktop Grid in such a way to minimize their *Turnaround Time*, that is defined as the time elapsing from the submission of a BoT to when its last task is completed. The turnaround time $TT(B_i)$ of a BoT B_i can be in turn decomposed in its *Waiting Time* $WT(B_i)$ (the time elapsing from its submission to when its first task is dispatched on a resource) and in its *Makespan* $MS(B_i)$ (the time elapsing from the beginning of the execution of its first task to the completion of the last one), that is:

$$TT(B_i) = WT(B_i) + MS(B_i) \tag{1}$$

As indicated by Eq.(1), turnaround time can be minimized by simultaneously minimizing both the waiting time and the makespan of individual BoTs. Achieving this simultaneous minimization for every BoTs B_i, however, may not possible in general, since the minimization of $MS(B_i)$ may adversely affect the makespan and the waiting time of other BoTs. More specifically, to minimize $MS(B_i)$ it is necessary to allocate to B_i at least one machine for each one of its tasks [1]. However, giving to B_i at a single time all the resources it requires may not leave enough resources to satisfy the needs of another BoT B_j, with the consequence that its makespan, its waiting time, or both may increase with respect to the case in which less resources had been given to B_i. As a matter of fact, if none of B_j's tasks has been scheduled yet, the assignment of too many resources to B_i may procrastinate the time instant at which B_j's first task will be dispatched, thus making $WT(B_j)$ increase. Furthermore, if B_j has not enough resources to run an instance for each of its tasks, $MS(B_j)$ may increase as well. Therefore, a sensible scheduling strategy must seek to strive a balance between allocating to each BoT enough resources to reduce its makespan, and to leave enough resources to other BoTs waiting in the queue to reduce their waiting time and makespan as well.

The problem of scheduling a set of independent BoTs can be, in essence, reduced to the problem of selecting one of the tasks waiting to be scheduled (*task selection*), an available machine (*machine selection*), and to dispatch the selected task on the chosen machine. Task selection requires in turn that one of the BoTs that still have to be completed is selected first (*BoT selection*), and that one of its tasks is selected and dispatched for execution (*individual BoT scheduling*). Accordingly, the scheduling algorithms

[1] In practice, however, the presence of heterogeneity and volatility requires – as discussed later – to use a certain degree of replication, so the number of resources that must be allocated to B_i to minimize its makespan is larger than that of its tasks.

we propose in this paper couple a BoT selection policy with a scheduling algorithm for individual BoTs. More precisely, we consider five different BoT selection policies (four of which are novel contributions of the present work), that are combined with a knowledge-free individual BoT scheduling algorithm (WQR-FT [16]), developed as part of our previous work, that – at the best of our knowledge – provides the best performance among similar algorithms. Therefore, in this paper we propose five different scheduling algorithms for multiple BoTs.

In the following subsections, after a description of the system model on which these algorithms are based (Sec. III-A), we will present the general structure of our scheduling algorithms (Sec. III-B) and the various BoT selection policies on which they are based (Sec. III-C).

A. System model

In our work we assume that Desktop Grids are composed by a set of independently-owned machines, connected by a public network (e.g., the Internet), that these machines may fail, or may be reclaimed by the respective owner at any time without any advance notice, and that they can reappear in the system after a variable and unknown amount of time. Thus, the status of each machine alternates between *available* and *unavailable*, and the time spent in the available state (the *time to failure*) and in the unavailable state (the *repair time*) are both assumed to be random variables.

We also assume that, in order to tolerate resource failures and departures, as well as to promote application performance [16], [27], [28], the Desktop Grid middleware provides a checkpointing-and-restart mechanism. More specifically, we assume that the middleware automatically takes checkpoints of running tasks by using *user-level* process checkpointing techniques [29] (e.g., [30], [31]), where a new checkpoint is automatically taken by code running in user space when specific system calls are invoked by tasks. Furthermore, we assume that these checkpoints are stored on a checkpoint storage system (e.g. [32], [33]) that enables all the machines in the Desktop Grid to access them.

Finally, we assume that scheduling is performed by a centralized scheduler, that receives all the BoT submissions and uses a separate queue for each BoT to hold its tasks that still have to be completed (*pending tasks*).

B. Scheduling Algorithms

As already anticipated, the scheduling algorithms proposed in this paper work by coupling a BoT selection policy with the same individual BoT scheduling algorithm. Consequently, the five scheduling algorithms proposed in this paper share a common structure, which is reported in Fig. III-B (where the notation summarized in Table I is used) in which BoT selection is performed first, and then individual BoT scheduling follows.

TABLE I

NOTATION USED IN THE SCHEDULING ALGORITHM DESCRIPTION

Symbol	Meaning
$\mathcal{M}$	The set of available machines
$M(t_i)$	Machine allocated to task t_i
$T(M_j)$	Identifier of the task allocated to machine M_j
$B(t_i)$	Identifier of the bag to which task t_i belongs
$Repl(t_i)$	The set of running replicas of task t_i
$Q(B_i)$	Scheduling queue associated with BoT B_i
$\mathcal{Q}$	The set of queues associated to submitted and uncompleted BoTs
$RepThresh$	The replication threshold

The pseudo-code listed in Fig. 1 encompasses a fixed part that schedules individual BoTs by means of WQR-FT, and a variable part performs BoT selection. A specific scheduling algorithm is obtained from the pseudo-code by instantiating it with one of the BoT selection policies discussed later.

The scheduler works in an event-driven way, that is it waits (line 3) for the occurrence of a scheduling-related event, that is an event that requires the scheduler to potentially take a scheduling decision, and then it performs BoT selection (line 21) and individual BoT scheduling (lines 22–28). The first scheduling-

```
1  M = {M_1, M_2, ..., M_N};
2  while true do
3      wait (event);
4      switch event do
5          case arrival of BoT B_i:
6              create queue Q(B_i); add Q(B_i) to Q;
7              for each task t_j ∈ B_i, insert t_j into Q(B_i);
8          case completion of replica r^x of task t_i:
9              add M(r^x) to M;
10             foreach task replica r^y ∈ Rep(t_i), r^y ≠ r^x do
11                 terminate r^y;
12                 add M(r^y) to M;
13             endfch
14             delete t_i from Q(B(t_i));
15             if (Q(B(t_i)) = ∅) then remove Q(B(t_i)) from Q;
16         case failure of machine M_i:
17             Repl(T(M_i)) –; remove M_i from the set of available machines;
18         case repair of machine M_i: add M_i to the set M of available machines;
19     endsw
20     while (Q ≠ ∅ and M ≠ ∅) do
21         B_j = Select_Bag();
22         CS(B_j) = {t_j ∈ Q(B_j) | Rep(t_j) < RepThresh};
23         next_task = t_N ∈ CS(B_j) such that ∀t_i ∈ CS(B_j), t_i ≠ t_N, Rep(t_N) < Rep(t_i);
24         starting_point(next_task) = Select_Best_Checkpoint(next_task);
25         M_i = Random_Select(M);
26         remove M_i from the set M of available machines;
27         dispatch(starting_point(next_task),M_i);
28         Repl(next_task)++;
29     endw
30 endw
```

Fig. 1. Pseudo-code of the scheduler

related event is the arrival of a new BoT B_i (lines 5–7), that is handled by creating a new queue $Q(B_i)$, in which all the tasks belonging to B_i are placed, that is added to the set $\mathcal{Q}$.

The next scheduling-related event is the successful termination of a *replica* of task t_i (lines 8–15). As will be discussed later, WQR-FT uses task replication to tolerate both poor scheduling decisions (due to the lack of any resource or task information) and the occurrence of machine faults. That is, it creates replicas of already-running tasks when there are available machines, and terminates all the running replicas of a task when the first one successfully terminates its execution. Therefore, when the first replica of a task terminates, all other replicas of the same task are immediately terminated by the scheduler (line 11), and the machine they used are placed in the set $\mathcal{M}$ of available machines (lines 9 and 12). Next, t_i

is removed from the queue $Q(B(t_i))$ of the corresponding BoT (line 14), since this task – having been completed – is no longer pending. Furthermore, if t_i was the last pending task in $Q(B(t_i))$, its termination implies also the termination of the whole BoT, that is handled by removing the corresponding task queue $Q(B(t_i))$ from the set $\mathcal{Q}$ of active BoT queues (line 15).

The last two scheduling-related event correspond to the failure and to the repair of a machine of the Desktop Grid. In case of failure of machine M_i (lines 16–17), the number of replicas of the corresponding task $T(M_i)$ is decremented by one (this action, as discussed later, will affect the decisions taken by WQR-FT) and M_i is removed from the set of available machines. Conversely, when a previously-failed machine M_i is repaired and reappears in the Desktop Grid, the only action that is performed is to add it to the set $\mathcal{M}$ (line 18).

When the processing of the just occurred event is completed, the scheduler checks (line 20) if there are tasks to be scheduled, i.e. that there are pending tasks ($\mathcal{Q} \neq \phi$) and available machines ($\mathcal{M} \neq \phi$). If there are no tasks to be scheduled, the scheduler goes back to line 3 to wait for the next event. If, instead, there is scheduling work to be done, the scheduler first selects the next BoT B_i from which the next task will be chosen by calling the Select_Bag() procedure (line 21) (that implements one of the BoT selection policies discussed in Sec. III-C), and then B_i is scheduled by means of the WQR-FT algorithm (lines 22–28).

WQR-FT is a replication-based scheduler, that is it creates replicas of already-running tasks when there are enough available resources. Although, as discussed later, replication is used to compensate the lack of information, it also provides fault tolerance since, in case of failure of a task replica, the other ones will continue their computation. WQR-FT works by keeping track of the number of running replicas of each task, and by always choosing the task that has the lowest number of running replicas (in case of tie, random selection is used). A *replication threshold (RepThresh)* – an upper limit on the number of running replicas per task – is set in order to avoid to waste too many resources (a replica that does not successfully terminate wastes the computing cycles used to run it). When a task of B_i has to be scheduled, WQR-FT builds the *candidate set* $CS(B_i)$ (line 22) – the set of tasks that have a number of running replicas strictly smaller than the replication threshold – and then selects the task in $CS(B_i)$ having the smallest number of running replicas (line 23). Then, the *best checkpoint* for the chosen task is selected (see below), one of the available machines is chosen at random (line 25) so that no knowledge about resource status is required), and the new task replica is started on it (line 28).

The concept of best checkpoint of a given task, that is defined as the checkpoint generated by the replica that has completed the largest part of its work, has been introduced in WQR-FT in order to further enhance application performance. Starting a new replica from the best checkpoint of the corresponding task (line 24), rather than from scratch, promotes indeed performance as the new replica leverages the computing work already performed by the one that produced the best checkpoint. It is worth to point out that this assumption does not limit the generality of our solution, as user-level checkpointing systems can be easily extended, as described below, to provide the best checkpoint selection capability. User-level checkpointing works by taking a checkpoint each time a task executes a system call so, since all the replicas of the same task execute the same code, the replica that at any given point in time has generated the largest number of checkpoints is the one that has executed the largest number of system calls, that corresponds to say that it has completed the largest amount of work. Therefore, in order to determine the best checkpoint for a given task, it is sufficient to keep track of the number of times that a checkpoint has been taken for each of its replicas.

C. BoT Selection Policies

The BoT selection policy has a direct effect on both the waiting time (since it determines when the first task of a BoT is scheduled) and the makespan (since it determines the number of machines allocated to each BoT). Therefore, it must be carefully crafted in order to be able at the same time to allocate enough resources to each BoT to reduce its makespan, and to distribute the available resources among all

the BoTs in order to reduce their waiting time. In this paper we propose the following set of five BoT selection policies, that differ in the way they attempt to achieve the above balance:

- *First Come First Served - Exclusive* (FCFS-Excl): This policy is a straightforward extension of the classical FCFS strategy used to schedule individual BoT applications on Desktop Grids, and consists in simply serializing the execution of the various BoTs according to the order of their arrival. More specifically, BoTs are scheduled in the order of their arrival, and the resources of the Desktop Grid are exclusively allocated to the currently running BoT (that is, no task of any other BoT is executed until the current one is completed). In order to fully exploit all the resources, the replication threshold is raised to a potentially unlimited value. This corresponds to say that – when there are no longer pending tasks for the current BoT – the machines that become free are kept busy by starting additional replicas of the tasks that are still running;
- *First Come First Served - Shared* (FCFS-Share): variant of *FCFS-Excl* in which the Desktop Grid is not exclusively allocated to a single BoT. As *FCFS-Excl*, BoTs are scheduled according to FCFS but, if the first BoT in FCFS order has no longer pending tasks, a machine that completes its task is allocated to the BoT that comes next. Therefore, as the number of completed tasks of the current BoT application increases, the number of resources allocated to the next BoT in the FCFS order increases as well;
- *Round Robin* (RR): in this policy, the various BoTs are scheduled in turn according to a circular order, starting from the first one in $\mathcal{Q}$ (the oldest BoT among the ones that still have to be completed) to the last one in $\mathcal{Q}$;
- *Round Robin - No Replica First* (RR-NRF): this policy is a variant of *RR* that gives priority to BoTs that do not have any task instance running. That is, when the scheduler is triggered, if there is a set $\mathcal{B}_x$ of BoTs that do not have any task replica running, the circular order of BoT selection is temporarily suspended, and BoTs in $\mathcal{B}_x$ are repeatedly selected until every BoT has at least a task running. Starting from that moment, circular BoT selection is restored;
- *Longest Idle* (LongIdle): this policy is motivated by the consideration that the turnaround time is often dominated by the waiting time, especially for high workload intensities. This policy attempts to reduce waiting time by giving preference to the BoT hosting the task that exhibits the largest waiting time, defined as the total amount of time in which the task has had no running replicas.

IV. Experimental evaluation

In order to asses the effectiveness of the proposed scheduling policies, we performed an exhaustive study, carried out by means a discrete-event simulator, in which we compared them for a large set of operational scenarios covering the whole landscape of Desktop Grid configurations and application workloads.

In our evaluation we compared the various scheduling algorithms in terms of the performance they deliver to BoTs, and their ability to maximize the utilization of the Desktop Grid resources. Delivered performance is measured by the average *Turnaround Time (TT)* of BoTs (defined as the arithmetic average of the turnaround time of individual BoTs – as defined in Eq.(1)). The ability of a scheduling policy to efficiently use resources is instead quantified by means of the *Relative Wasted Time (RWT)*, defined as the fraction of computation time wasted to run *useless* replicas. A replica is considered useless if it is terminated by the scheduler (as consequence of the successful completion of the first replica of the same task), or because of a machine crash, without having ever produced a checkpoint better than the one stored for the corresponding task.

In the rest of this section we describe the Desktop Grid configurations first (Section IV-A), we continue with the description of the workloads (Section IV-B), and then we conclude with the results obtained in our simulation experiments (Section IV-C).

A. Desktop Grid configurations

Generally speaking, Desktop Grids differ from each other in terms of the heterogeneity and the availability of their resources. In order to ensure the generality of our results, in this study we defined a set

of six Desktop Grid configurations, obtained by combining two heterogeneity levels with three availability values, in such a way to cover a large set of real systems.

1) Resource Heterogeneity: We quantify heterogeneity in terms of the distribution of the computing power delivered by individual resources of the Desktop Grid. The computing power P_i of machine i is represented as a real number whose value is directly proportional to the speed of the machine (i.e., a machine i with $P_i = 10$ is twice faster than a machine j with $P_j = 5$). The *total computing power* P of a Desktop Grid is defined as the sum of the computing power of individual machines, that is $P = \sum_{i=1}^{|\mathcal{M}|} P_i$ (where $\mathcal{M}$ denotes the set of machines of the Desktop Grid).

The heterogeneity levels we used for our experiments have been chosen in such a way to be representative of two classes of Desktop Grids placed at the opposite ends of the spectrum, namely those whose resources have identical computing power (i.e., henceforth referred to as *homogeneous* configurations) and those whose resources exhibit relatively high differences in their computing power (henceforth referred to as *heterogeneous* configurations).

Since, to the best of our knowledge, the literature lacks a common agreement on the computing power distributions of typical Desktop Grids, we decided – as in [24] – to set the total computing power $P = 1,000$ (for all the configurations used in the experiments) of the Desktop Grid and to progressively add machines – whose computing powers were drawn from a specific probability distribution – until the sum of their computing power matched the value of P. Therefore, the number $|\mathcal{M}|$ of machines included in a given configuration depends on the particular computing power distribution chosen for that configuration.

For homogeneous configurations, the computing power was fixed to 10 for all machines, (i.e., $P_i = 10, i = 1, \ldots, |\mathcal{M}|$), that resulted in a system composed by 100 machines. For heterogeneous configurations, the computing power was assumed to be uniformly distributed in the [2.3,17.7] interval, resulting in an average value of 10 and a variance of 19.76, that gave rise to a configuration including 93 machines. This specific distribution was chosen, as discussed in [24], to reproduce a computing power distribution complying with the Moore's Law (that states that computer power double every 18 months) under the hypothesis that the construction dates of the machines in the Desktop Grid span a 5 years interval. Under these assumptions, we can indeed expect that the fastest machine of the Desktop Grid is 8 times faster than the slowest one, and this corresponds to the limits of the [2.3,17.7] interval.

2) Resource Availability: The availability of a Desktop Grid configuration is quantified in terms of its *average machine availability* $\overline{A}$, that is defined as:

$$\overline{A} = \frac{1}{|\mathcal{M}|} \sum_{i=1}^{|\mathcal{M}|} A_i \tag{2}$$

where A_i is the asymptotic availability of machine M_i, a percentage value that quantifies the fraction of time during which M_i can be used to run Desktop Grid applications, that is in turn defined as:

$$A_i = \frac{MTTF_i}{(MTTF_i + MTTR_i)} \tag{3}$$

where $MTTF_i$ and $MTTR_i$ denote the mean of the distributions of the time to failure and of repair times of M_i, respectively.

In order to consider various Desktop Grid configurations, we used three different availability values. Enterprise Desktop Grids, being characterized by a high resource stability, were represented by configurations whose average availability was set to 98% (henceforth referred to as *High-Avail* configurations). Conversely, Volunteer Desktop Grids were represented by configurations whose availability was set to 50% (*Low-Avail* configurations), since in these platforms the participating machines come and go unpredictably with a relatively high-frequency. A third availability value (75%) was used in order to consider also

Desktop Grid configurations that could be placed in between Enterprise and Volunteer Desktop Grids (*Medium-Avail* configurations).

To obtain these values, we proceeded as follows. As indicated by Eqs.(2) and (3), a given average availability $\overline{A}$ can be obtained by suitably setting $MTTF_i$ and $MTTR_i$ ($i = 1, \ldots, |\mathcal{M}|$), that in turn requires the knowledge of the distributions of fault and repair times, as well as of their parameters, for all the machines in $\mathcal{M}$. In accordance with the studies reported in [34], [35], we used the Weibull distribution for fault times, while for the repair times (for which, at the best of our knowledge, no results are published in the literature) we chose the normal distribution. For each of the three availability levels, the parameters of the repair times distributions were set to the same value for all the machines (i.e., $MTTR_i = MTTR_j, i, j = 1, \ldots, N$). For the parameters of the Weibull distribution (the *shape* and the *scale*) characterizing fault times, we took instead a different approach in which we partitioned the machines in 15 different groups, each one associated with specific values of shape and scale (obtained from the results published in [35] and reported in Table II), that were used for all the machines in the same group.

TABLE II

MTTF AND MTTR FOR THE MACHINE GROUPS

Machine	**High-Avail**		**Medium-Avail**		**Low-Avail**	
Group	MTTF (sec.)	MTTR (sec.)	MTTF (sec.)	MTTR (sec.)	MTTF (sec.)	MTTR (sec.)
1	773119	1800	23193.60	5400	7731.19	5400
2	1044610		31338.40		10446.10	
3	997908		29937.20		9979.08	
4	816990		24509.70		8169.90	
5	330479		9914.37		3304.79	
6	1288810		38664.20		12888.10	
7	426508		12795.20		4265.08	
8	487921		14637.60		4879.21	
9	779938		23398.20		7799.38	
10	997908		29937.20		9979.08	
11	600641		18019.20		6006.41	
12	331339		9940.17		3313.39	
13	315787		9473.60		3157.87	
14	319848		9595.44		3198.48	
15	407545		12226.40		4075.45	

The three availability values used in our study correspond to the MTTF and MTTR values reported in Table II. More specifically, the MTTF values for the *High-Avail* case have been directly derived from [35], while the other ones have been computed by reducing these values to 3% (*Medium-Avail*) and 1% (*Low-Avail*), respectively. Furthermore, the mean and variance of the repair time for the *High-Avail* case were set to 1800 sec. and 300 sec., respectively, while for the other two cases we set them to 5400 sec. and 800 sec., respectively.

B. Workloads

For our study, we consider various workloads, each one consisting in a set of BoTs that arrive to the scheduler at a certain rate and require a given amount of work. Both the amount of work required by each task (*task execution time*), and the time elapsing between two consecutive BoT arrivals (*BoT interarrival time*), are assumed to be random variables distributed according to specific distributions. We assume that BoT interarrival times were exponentially distributed with rate λ (henceforth referred to as *BoT arrival rate*). Task execution times are assumed to be uniformly distributed in the interval $[X - 50\%, X + 50\%]$,

where X is the *granularity* of tasks, that is defined as their mean execution times referred to a *baseline machine* having computing power $P = 1$ [2]. We consider BoT that include tasks characterized by different granularities by defining a set of *task classes*, each one corresponding to a given granularity values, and by defining a set of *workload mixes*, each one corresponding to a specific combination of the basic task classes. The workload we use for our study are obtained by combining a value for the BoT interarrival rate with a specific workload mix. Therefore, all the BoTs generated when using a given workload are characterized by the same workload mix.

Four our study, we used 8 task mixes and 3 BoT arrival rates – described below – that give rise to 24 different workloads.

1) Workload mixes: As already anticipated, each workload mix corresponds to a particular combination of task classes, each one characterized by a single value of task granularity. We define four basic task classes, named *Very Small*, *Small*, *Medium*, and *Large*, corresponding to granularity of 1,000 sec. (*Very Small*), 5,000 sec. (*Small*), 25,000 sec. (*Medium*), and 125,000 sec. (*Large*), and we use them to obtain 8 different task mixes, each one corresponding to a particular distribution of task classes, expressed in terms of the percentage of tasks of the various classes that belong to individual BoTs. These percentages are reported, for all the task mixes, in Table III (where P_{VS}, P_S, P_M, and P_L denote the fraction of very small, small, medium, and large tasks). As can be observed by Table III, workload mixes can be divided in two

TABLE III
WORKLOAD MIXES

Mix group	**Task mix**	P_{VS}	P_S	P_M	P_L
Single class	*All_VS*	100%	0%	0%	0%
	All_S	0%	100%	0%	0%
	All_M	0%	0%	100%	0%
	All_L	0%	0%	0%	100%
Multiple class	*Uniform*	25%	25%	25%	25%
	Short	50%	$16.\overline{3}$ %	16.3 %	$16.\overline{3}$ %
	Med	$16.\overline{3}$%	$16.\overline{3}$%	50%	$16.\overline{3}$%
	Long	16.3%	16.3%	16.3%	50%

distinct groups. The first group (henceforth referred to as *single class* mixes) included mixes characterized by a single task class (that is, all the BoTs submitted to the Desktop Grids are composed by tasks having the same granularity), and is made up by *All_VS*, *All_S*, *All_M*, and *All_L*. The second group (henceforth referred to as *multiple class* mixes) includes instead mixes characterized by multiple task classes, and is made up by *Uniform*, *Short*, *Med*, *Long*.

Given the fraction of tasks belonging to a each class, the distribution of the task execution times of BoTs for a given workload is given by:

$$P_{VS} * U(500, 1500) + P_S * U(2500, 7500) + P_M * U(12500, 37500) + P_L * U(62500, 187500) \quad (4)$$

where P_{VS}, P_S, P_M, P_L are set according to Table III, and $U(a, b)$ denotes the uniform distribution with parameter a and b. Eq. (4) is used to generate the various BoTs composing a given workload in the following way. First, we fix its *application size* AS of a BoT (defined as the sum of the execution times of its constituent tasks), and then we repeatedly generate new tasks until the sum of their execution times, computed by drawing random values from the distribution in Eq.(4), matches AS. For this study, AS was set to 3,600,000 sec. for all the BoTs regardless of their specific task mix, (as done also in [24]), since this allowed us to evaluate the impact not only of resource and task heterogeneity, but also of task-to-machine ratio. As a matter of fact, since both the application size and the number of machines is constant for a

[2]It is worth to point out that the actual execution time of a task depends on the computing power of the machine on which it will be executed. In our case, given that the average computing power of machines equal to 10, a task whose granularity is 125,000 sec. will be executed – on average – in 12,500 sec.

given heterogeneity level, varying the task granularity corresponds to change the average number of tasks per BoT executed by each machine.

2) BoT arrival rates: We considered a set of different arrival rates chosen in such a way to reproduce various load conditions on the resources of the Desktop Grid. We quantify the amount of work induced on the Desktop Grid by a given workload by means of its *load factor* L [36], a real quantity representing the proportion of time that the Desktop Grid is busy defined as:

$$L = \frac{\sum_{i=1}^{|\mathcal{M}|} B_i}{\sum_{i=1}^{|\mathcal{M}|} (B_i + I_i)} \tag{5}$$

where B_i (the *busy time*) denotes the amount of time that machine M_i spends processing tasks of the various BoTs, and I_i (the *idle time*) denotes the time that M_i spends idle waiting for tasks to execute. In order to evaluate the various scheduling algorithms under various load conditions, in our study we considered workloads yielding load factors of 0.5, 0.75, and 0.95, named as *low-intensity*, *medium-intensity*, and *high-intensity*, and denoted as L_{low}, L_{med}, and L_{hi}, respectively [3].

As shown in [37], a given load factor $L_X, (X \in \{low, med, hi\})$ depends on the BoT arrival rate λ_X and the *CPU occupancy time* C_X (the total amount of time requested by all the tasks in a singe BoT), that is:

$$L = \lambda \cdot C \tag{6}$$

For our study, we used the same value of C for all our experiments, that is $C_X = C, X \in \{low, med, hi\}$. Consequently, we obtained a workload characterized by a given load factor L_X by setting the interarrival rate λ_X for its BoTs as:

$$\lambda_X = \frac{L_X}{C} \tag{7}$$

We compute the CPU occupancy time C for a given workload as:

$$C = \frac{AS}{\sum_{i=1}^{|\mathcal{M}|} EP_i} \tag{8}$$

where EP_i is the *effective computing power* of machine M_i, that is the computing power that M_i delivers over any time interval during which the machine may also be unavailable, which is in turn defined as:

$$EP_i = P_i \cdot A_i \tag{9}$$

where P_i and A_i denote the computing power and the asymptotic availability of machine M_i, respectively. Intuitively, Eq.(9) states that the effective computing power delivered by a machine M_i whose availability is A_i is equivalent to that of a machine M_j that is fully available but delivering a computing power $P_j = P_i \cdot A_i$. For example, a machine whose computing power and availability are 10 and 0.5, respectively, is considered to be equivalent to a machine with a computing power of 0.5 that is 100% available. By looking at Eqs.(8) and (9), we note that the CPU occupancy time does not depend on the particular workload, since AS assumes the same value for all the workloads, but depends instead on the characteristics of the Desktop Grid configuration (and, precisely, on the distribution of the computing power and of availability of individual machines). The consequence is that, for different configurations, the same load factor will correspond to a different value of λ, as reported in Table IV.

[3]Note that in order for the system to be in a stable state (that is, a state in which the turnaround time of BoTs does not grow infinitely) for a given workload, we must have $L < 1$.

TABLE IV

BoT INTERARRIVAL RATES FOR THE VARIOUS SCENARIOS

Desktop Grid configuration		$\sum_{i=1}^{N} EP_i$	C	λ		
				L_{low}	L_{med}	L_{hi}
High-Avail	Homogeneous	996.68	3612	0.00014	0.00021	0.00026
High-Avail	Heterogeneous	997.89	3607.61	0.00014	0.00021	0.00026
Medium-Avail	Homogeneous	759.71	4738.68	0.00011	0.00016	0.00020
Medium-Avail	Heterogeneous	763.82	4713.17	0.00011	0.00016	0.00020
Low-Avail	Homogeneous	526.41	6838.81	0.00007	0.00011	0.00014
Low-Avail	Heterogeneous	531.87	6768.52	0.00007	0.00011	0.00014

As a final consideration, we note that Eq.(8) does not take into account the overheads due to task replications and resubmissions, as well as to checkpointing, that were hard to express in a closed formula. Furthermore, it is based on the simplifying assumption that the workload is perfectly divisible (i.e., any machine can receive any arbitrarily small amount of work), while in practice it is not, since the work is discretized into individual tasks. However, these simplifications do not hinder the generality of the results. As a matter of fact, the value of L computed by means of Eq.(8) corresponds to a lower bound for the actual load factor (e.g., a value $L = 0.75$ indicates that the actual load factor is *at least* 75%), and if a given result holds for the lower bound, it also holds for a (slightly) higher value of the load factor.

C. Results

Let us discuss now the results we obtained by performing simulation experiments for all the combinations of the scenarios and workloads described before. In all the experiments, we set the replication threshold of WQR-FT to 2 (i.e., the scheduler attempts to always have two running replicas per task) since, as shown in [16], higher replication thresholds bring negligible performance benefits at the price of a much higher overhead caused by the larger number of replicas per task. The only exception to this rule has been made for the *FCFS-Excl*, where (as already mentioned) an unlimited replication threshold has been used. Finally, for all the Desktop Grid configurations, we assumed that the time taken to transfer (retrieve) a checkpoint file to (from) the server was uniformly distributed in the interval [240,720] seconds.

As mentioned before, our primary performance indices are the Average Turnaround Time (*AVT*) of BoTs, and the Relative Wasted Time (*RWT*). In addition to these indices, we have also collected other information that help us to explain the phenomena that yield to the measured values for AVT and RWT. More specifically, we collect:

- the *Average Makespan Time*, defined as the arithmetic average of the makespan of individual BoTs;
- the *Average Bag Waiting Time*, defined as the arithmetic average of the waiting time of individual BoTs;

For all the above performance indices, we computed 95% confidence intervals with a relative error of 2.5% or less.

In the rest of this section, we will first discuss the results obtained for workloads characterized by single class mixes (Sec. IV-C.1), and then those corresponding to workloads characterized by multiple class task mixes (Sec. IV-C.2).

1) Single Class Workloads: Single class workloads are those corresponding to the *All_VS*, *All_S*, *All_M*, and *All_L* workload mixes (see Table III).

The results corresponding to Desktop Grid configurations characterized by a high availability value (representing, as already mentioned, Enterprise Desktop Grids), are reported from Fig. 2 through Fig. 5.

In the case of homogeneous configurations and low-intensity workloads (Fig. 2), we note that, for workload characterized by very small and small task granularities (the bar groups labeled as *All_VS* and

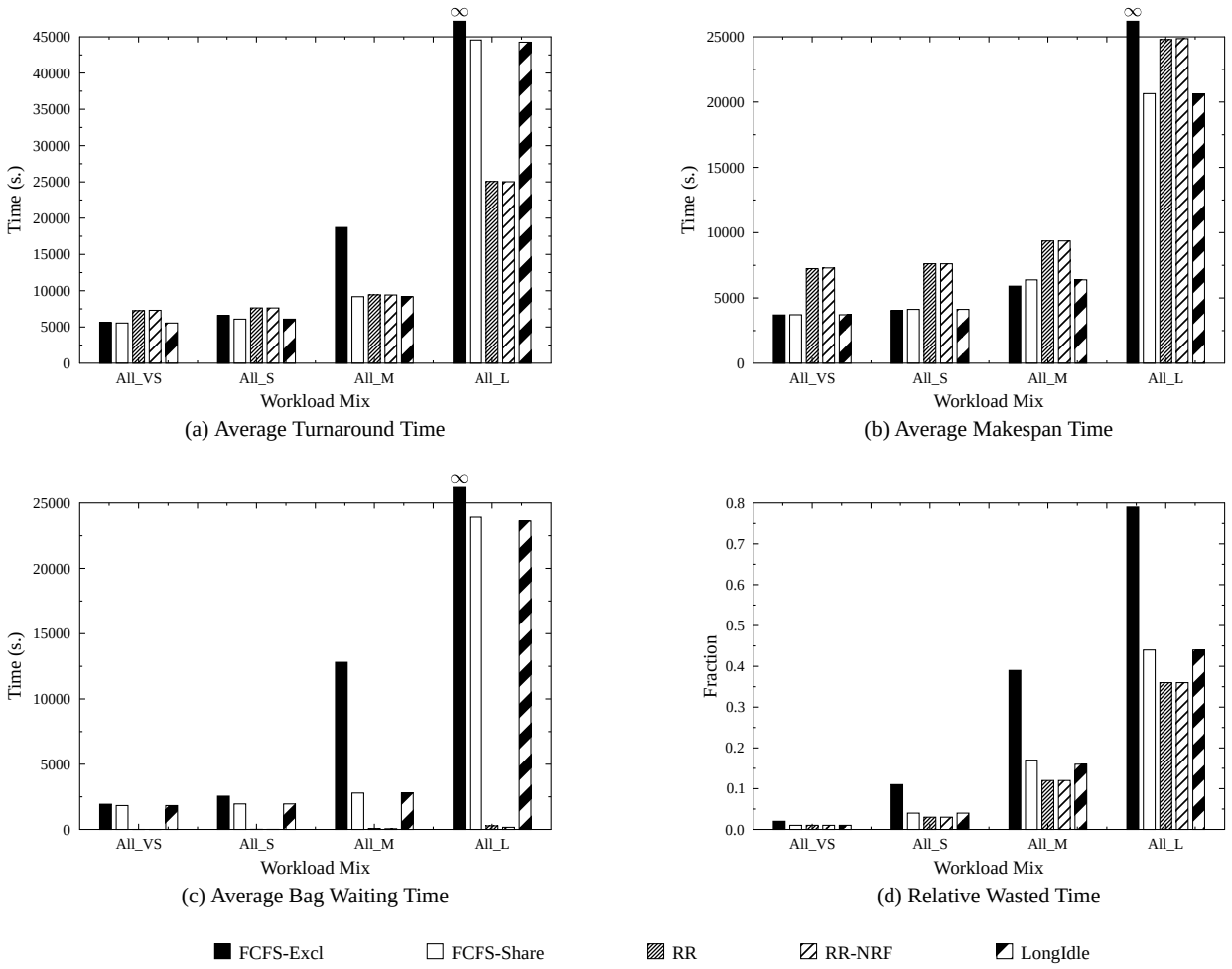

Fig. 2. Results for the High-Avail/Homogeneous Desktop Grid configuration and Single-Class/Low-Intensity workloads

All_S, respectively), RR and RR-NRF perform slightly worse than the other ones (Fig. 2(a)). The results reported in Fig. 2(b) and Fig. 2(c) indicate that this performance gap is not due to higher values of the waiting time or of the RWT (that are negligible for both RR and RR-NRF), but – as indicated by the results in Fig. 2(b)– it is instead due to higher values of the makespan. This depends on the fact that, for these granularity values, each BoT application has, on average, a number of tasks much larger than that of available resources ($\sim$ 36 tasks/machine for *All_VS*, and $\sim$ 7 tasks/machine for *All_S*). Therefore, strategies that tend to allocate a larger amount of resources to fewer BoTs perform better than those that simultaneously allocate fewer resources to a larger number of BoTs, since the larger the number of resources allocated to each bag, the lower the corresponding makespan. Furthermore, given the short average duration of individual tasks (1,000 sec. and 5,000 sec., respectively), the waiting time is not an issue, so strategies like RR and RR-NRF, that aim at minimizing the waiting time, do not obtain any appreciable benefit with respect to the other strategies that instead focus on reducing the makespan.

However, by looking at the average turnaround time (Fig. 2(a)) for workload mixes characterized by higher granularity values (corresponding to the bar groups labeled as *All_M* and *All_L*), we observe that the performance of RR and RR-NRF are comparable to (*All_M*) and much better than (*All_L*) those of the other strategies. Furthermore, it is worth to point out that for the *All_L* workload mix FCFS-Excl makes the makespan grow to an unlimited value, as graphically indicated by the fact that the corresponding bar goes beyond the border of the graph. The inspection of the results concerning the average makespan

(Fig. 2(b)) and the waiting time (Fig. 2(c)) provides again the explanation for this behavior. As a matter of the fact, as indicated in Fig. 2(c), the above performance gap is due to the waiting time, rather than to the makespan that, for RR and RR-NRF, is even higher than the other strategies (see Fig. 2(b)). This depends on the fact that, for these workload mixes, the average number of tasks per BoT is comparable to or smaller than that of the machines of the Desktop Grid (~ 1.5 tasks/machine for *All_M* and ~ 0.3 tasks/machine for *All_L*), so there are always enough machines to simultaneously schedule several BoTs, thus significantly reducing their waiting times (see Fig. 2(c)). Conversely, FCFS-Excl – by allocating all the resources to a single BoT application even in these situations where this is not necessary – makes the turnaround time grow to an unlimited value as consequence of the simultaneous growth of the makespan and of the waiting time. FCFS-Share and LongIdle perform worse than RR and RR-NRF as well because of their higher values for the average waiting time. An interesting observation that can be made by looking at the results for FCFS-Share and LongIdle is that their exhibit practically identical performance. This is due to the fact that, when the currently running BoT has still pending tasks without replicas, they behave exactly in the same way. As a matter of fact, in this case the pending tasks of a given BoT will always exhibit a larger amount of idle time with respect to tasks belonging to BoTs submitted later. LongIdle will instead start choosing BoTs different from the oldest one only when all its tasks have at least a replica running.

The other performance metric used for our comparison, the RWT (Fig. 2(d)), tells a different story. As a matter of fact, RR and RR-NRF are always able to outperform the other strategies for all the workload mixes. This is the direct consequence of the fact that, by using a circular order for bag selection, they start replicate much later than the other strategies. However, in this particular configuration, characterized by homogeneous resources and high availability, the performance benefits of replication are marginal since the vast majority of replicas is useless. As a matter of fact, since the machines are identical and their availability is high, the replica of a given task that is started first is always ahead of the other ones of that task, thus making them unable to generate checkpoints better than those produced by the first replica.

When the machines of the Desktop Grid are heterogeneous (but still highly-available), the results for low-intensity workloads (shown in Fig. 3), are similar to those obtained for the homogeneous configuration. As in the latter case, RR and RR-NRF perform slightly worse than the other strategies for the *All_VS* and *All_S* workload mixes, and comparably to (better than) them for the *All_M* (*All_L*) mixes. The only notable difference with respect to the homogeneous case is that in this scenario – for the higher granularity workload mixes – the performance gap between RR and RR-NRF and the other ones is smaller (that is, they perform slightly worse, while the other ones slightly better). This is due to the fact that when resources are heterogeneous, replication pays off, as it allows to better tolerate poor scheduling decisions due to the lack of resource information. Therefore, FCFS-Excl, FCFS-Share and LongIdle, being more prone to replication – especially for *All_M* and *All_L* – than RR and RR-NRF, exhibit performance better than in the homogeneous case, while the opposite is true for RR and RR-NRF. However, despite this difference in performance, also in this case RR and RR-NRF are clear winner in terms of their ability of profitably exploit resources, as indicated by the results concerning the RWT (see Fig. 3(d)).

Similar considerations can be made for these Desktop Grid configurations when the workload intensity is high (see Figs. 4 and 5). As for low intensity workloads, RR and RR-NRF perform slightly worse than the other scheduling algorithms for the *All_VS* and *All_S* workload mixes, and better for *All_M* and *All_L* (see Figs. 4(a) and 5(a)). However, the performance gap between RR and RR-NRF, and the other strategies, is now much higher than in the low-intensity case, since FCFS-Excl, FCFS-Share, and LongIdle fail to yield a finite value for the turnaround time since, as can be seen from Figs. 4(c) and 5(c), their waiting time is practically unbounded, while this is not the case of RR and RR-NRF. Furthermore, as in the low-intensity case, RR and RR-NRF result in much lower values for RWT.

When the Desktop Grid availability decreases to 75% (*Medium-Avail*), our results (that have been omitted from this paper because of space constraintsRR and RR-NRF perform slightly worse than the other ones for the *All_VS* and *All_S* workload mixes, and comparably to (much better than) for *All_M* (*All_L*). Furthermore, in this latter case FCFS-Excl, FCFS-Share and LongIdle yield unbounded values

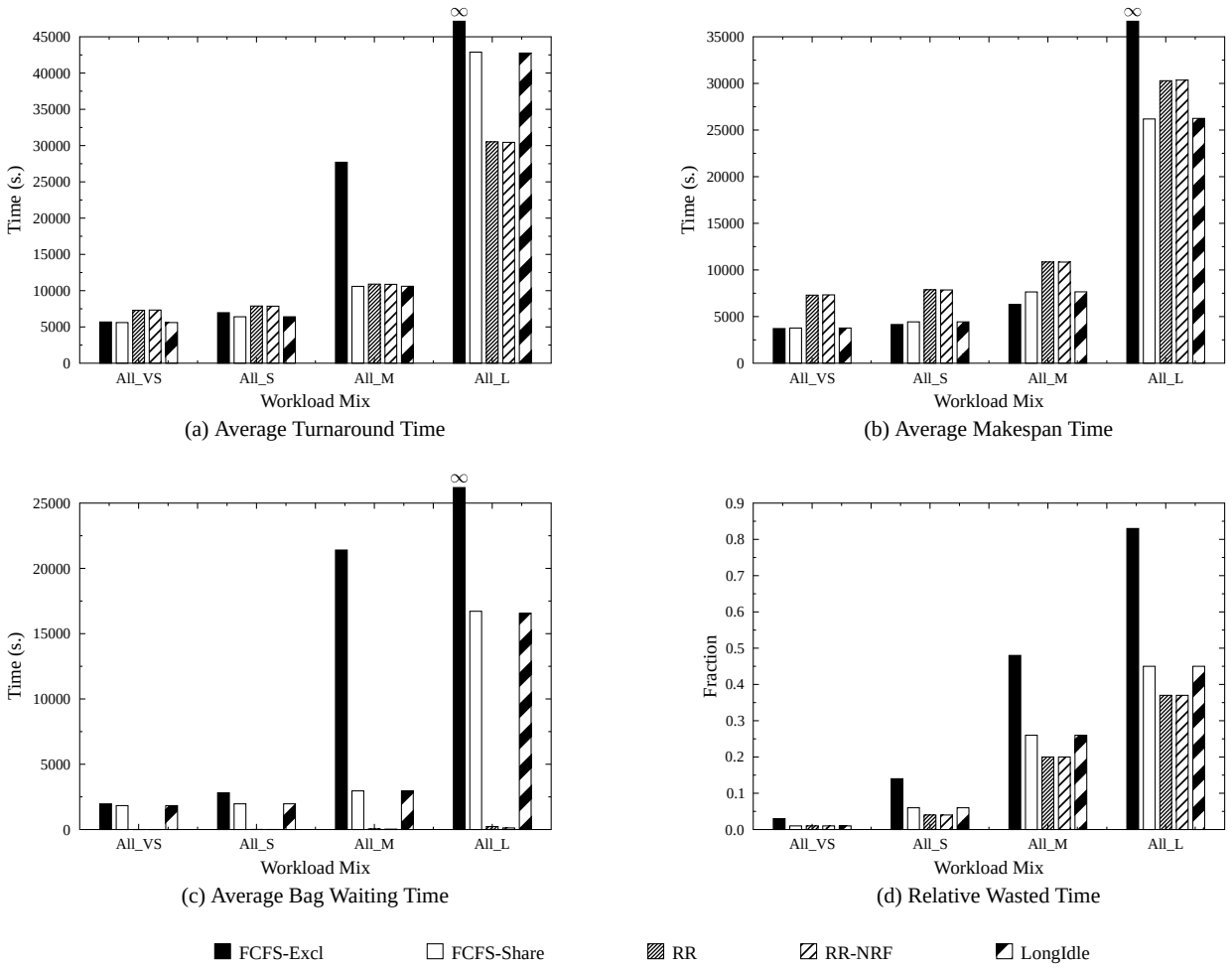

Fig. 3. Results for the High-Avail/Heterogeneous Desktop Grid configuration and Single-Class/Low-Intensity workloads

of the turnaround time also for L_{low} workloads, and not only for L_{hi} ones. The results for the RWT, again, show a large advantage of RR and RR-NRF, that exhibit performance gains as large as 10% for FCFS-Share and LongIdle and 25% for FCFS-Excl (homogeneous configurations), and slightly lower for the heterogeneous configurations since, as already discussed, creating an higher number of task replicas pays off when the machines are heterogeneous.

Finally, let us consider the results obtained for low availability configurations (corresponding, as already mentioned, to Volunteer Computing Desktop Grids), that are shown from Fig. 6 through 9. Also for these configurations, in the case of low intensity workloads, the strategies exhibit the same behavior observed for high and medium availability values. That is, for low granularity workload mixes (*All_VS* and *All_S*), RR and RR-NRF perform slightly worse than the other strategies, while their turnaround time is better for the other workload mixes. However, in this case, we observe that for high-intensity workloads all the scheduling strategies fail to yield a bounded turnaround time regardless of the heterogeneity of the platform. This, however, is not unexpected, since a 0.90 load factor make the system work under saturation, and there is no room to create enough replicas to tolerate poor scheduling decisions or machine failures.

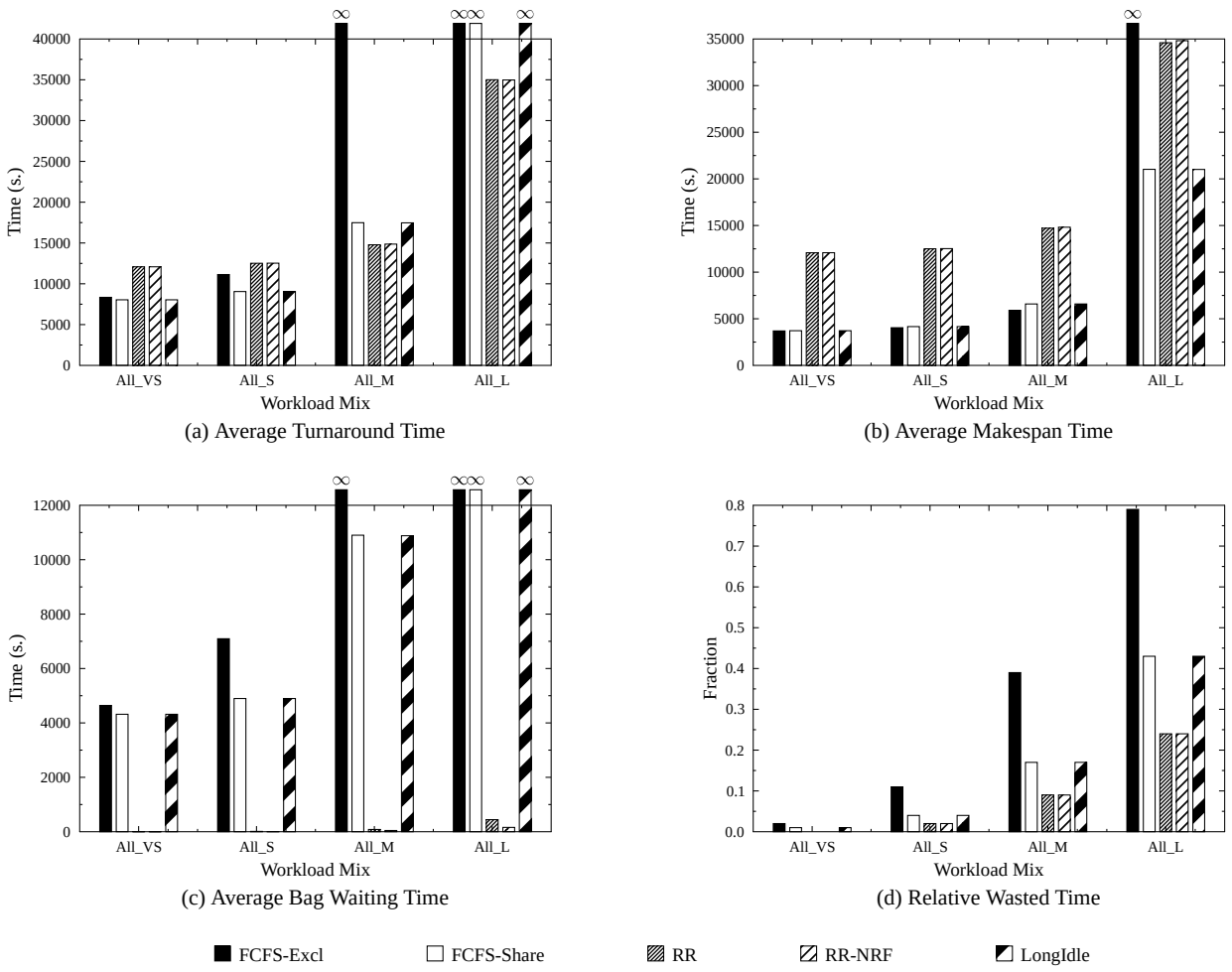

Fig. 4. Results for the High-Avail/Homogeneous Desktop Grid configuration Single-Class/High-Intensity workloads

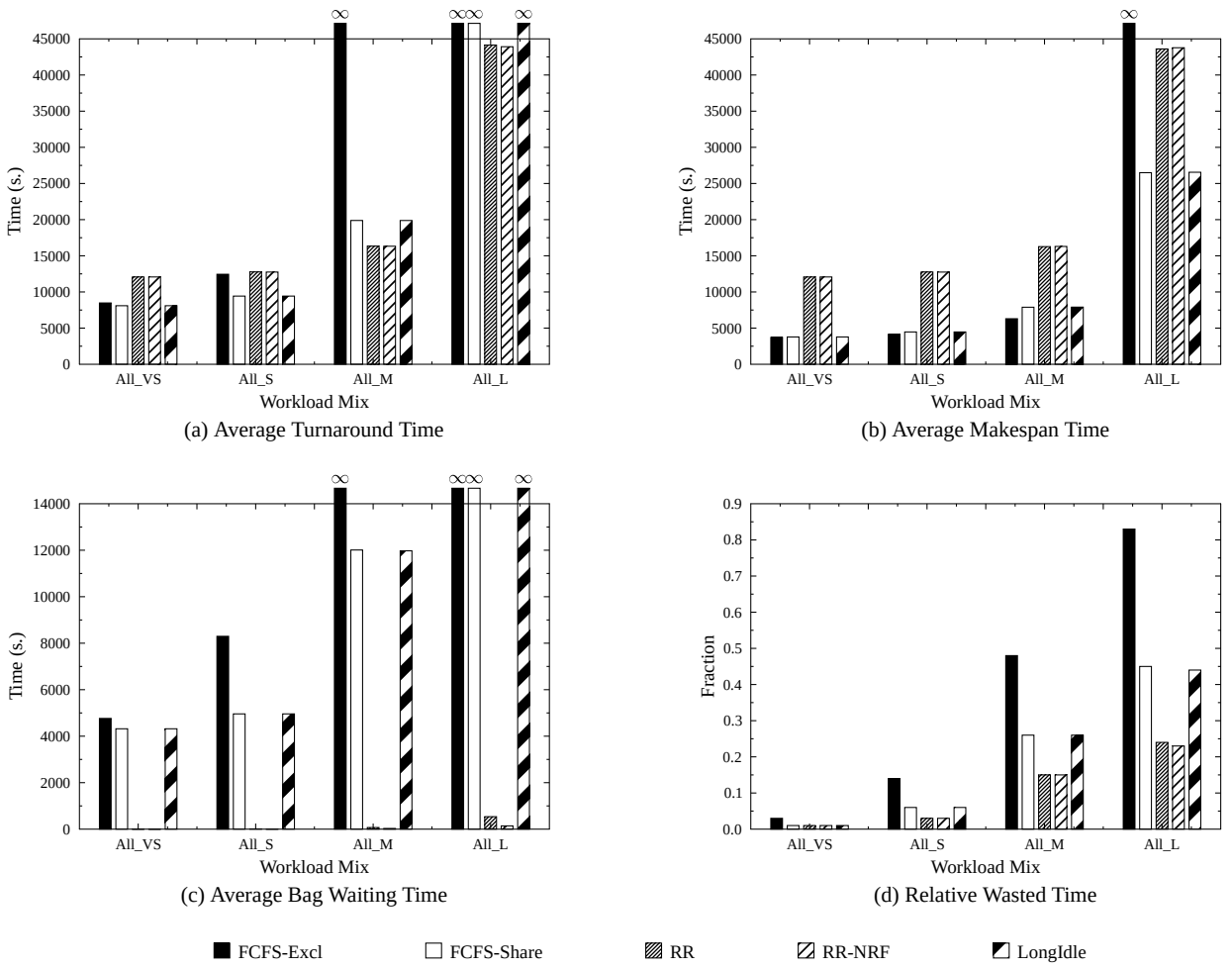

Fig. 5. Results for the High-Avail/Heterogeneous Desktop Grid configuration and Single-Class/High-Intensity workloads

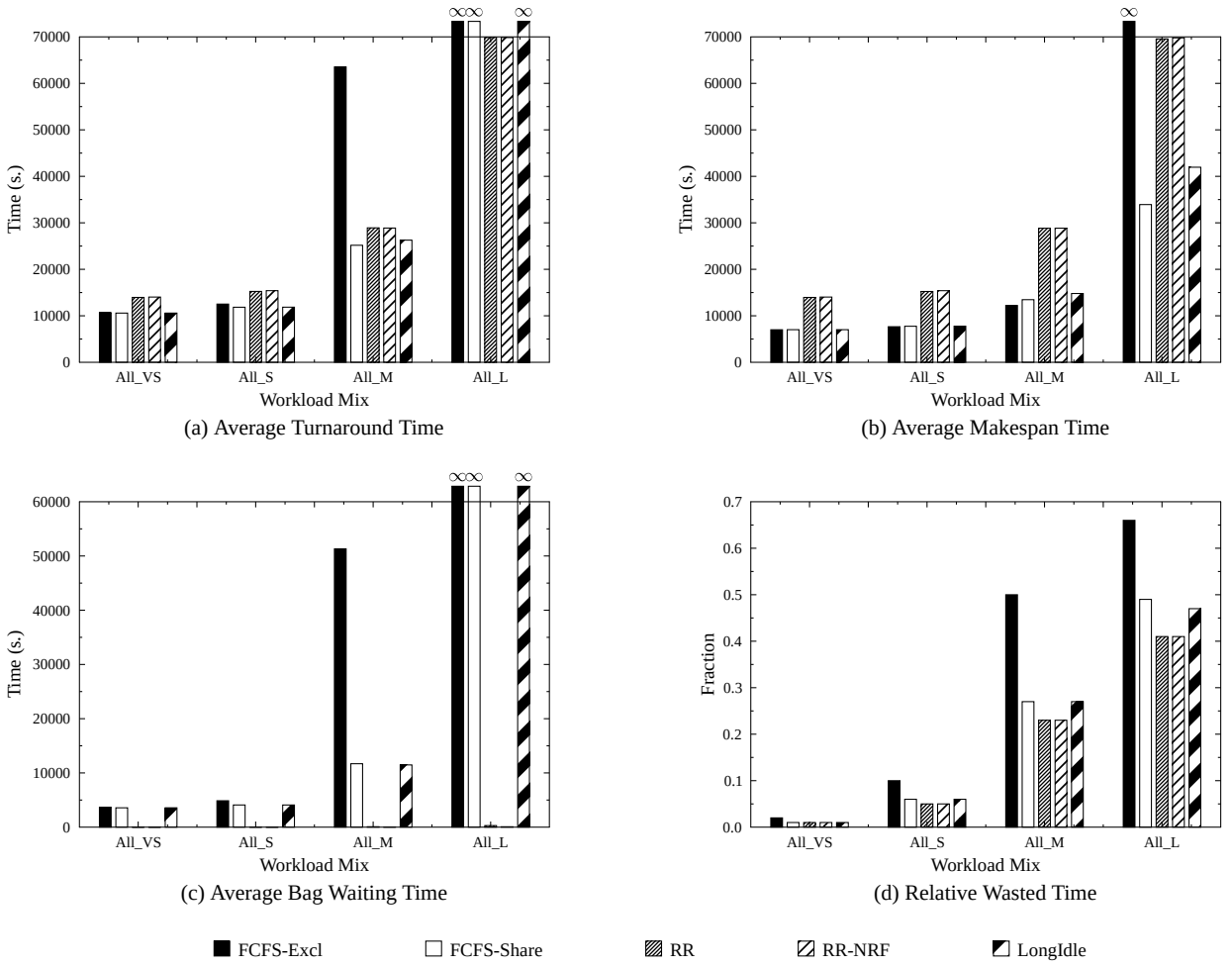

Fig. 6. Results for the Low-Avail/Homogeneous Desktop Grid configuration and Single-Class/Low-Intensity workloads

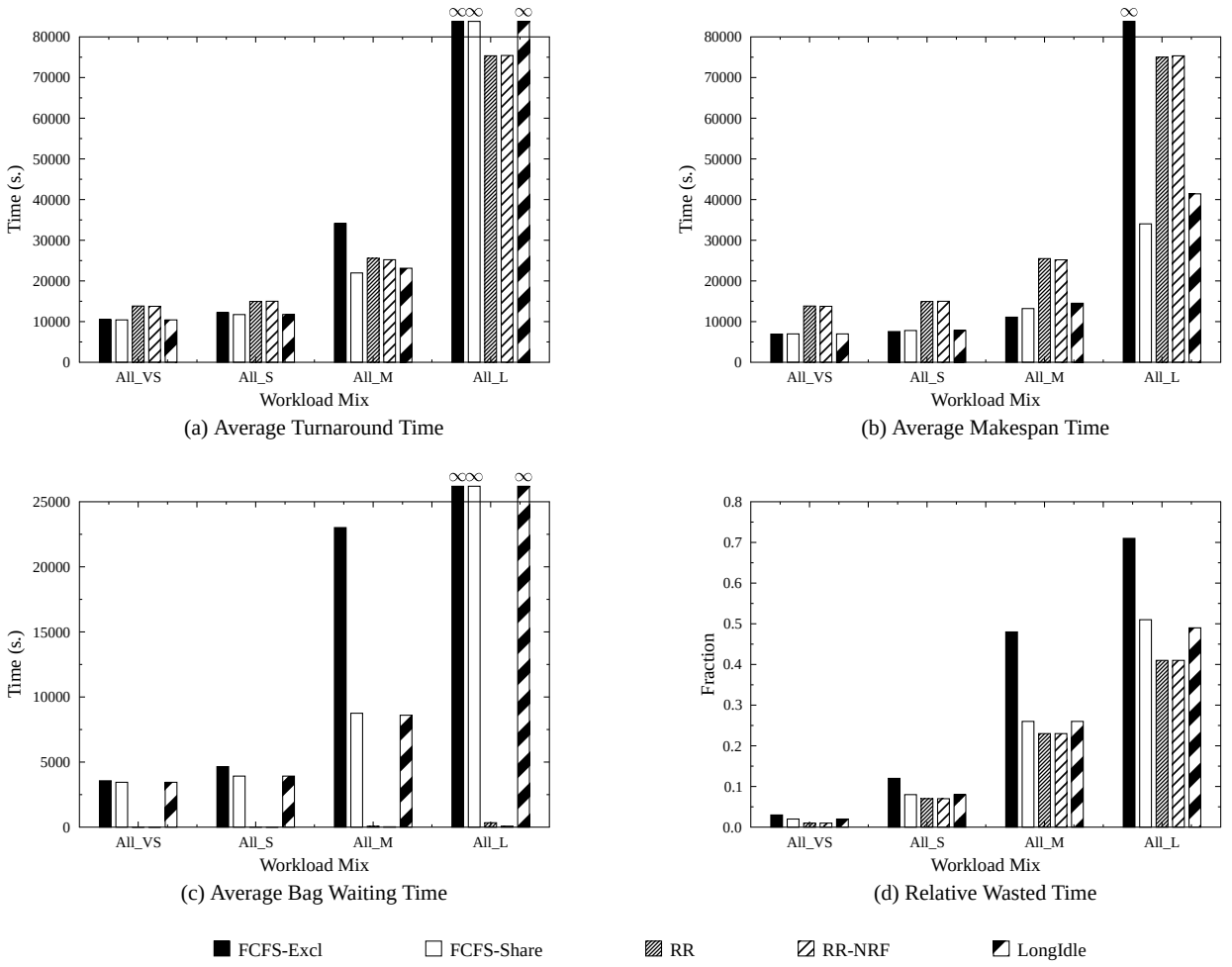

Fig. 7. Results for the Low-Avail/Heterogeneous Desktop Grid configuration and Single-Class/Low-Intensity workloads

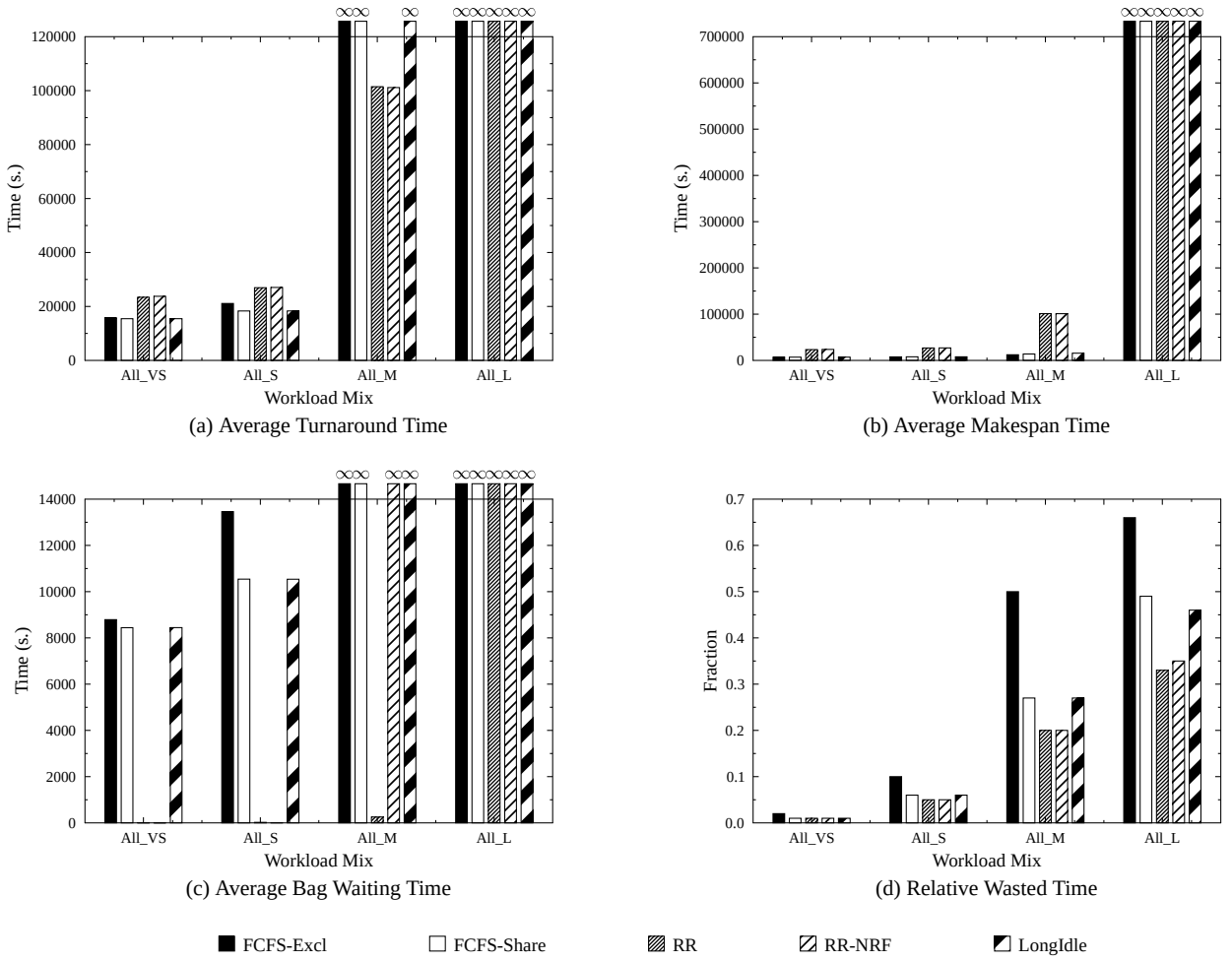

Fig. 8. Results for the Low-Avail/Homogeneous Desktop Grid configuration and Single-Class/High-Intensity workloads

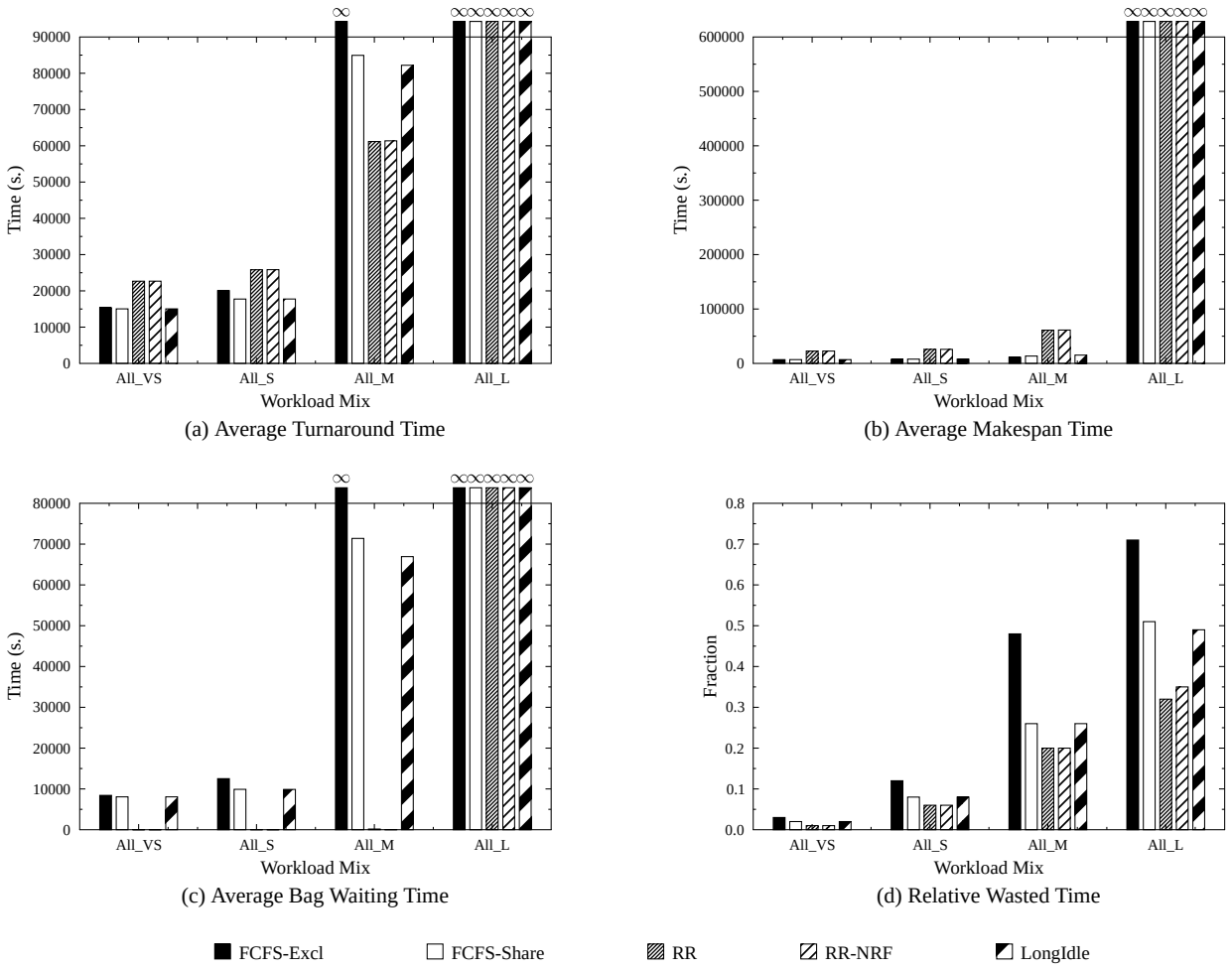

Fig. 9. Results for the Low-Avail/Heterogeneous Desktop Grid configuration and Single-Class/High-Intensity workloads

2) Multiple Class Workloads: Multiple class workloads are those characterized by the *Uniform*, *Short*, *Medium* and *Long* workload mixes (see Table III).

The results corresponding to Desktop Grid configurations characterized by high availability values are reported from Fig. 10 through Fig. 13. For homogeneous configurations and low-intensity workloads

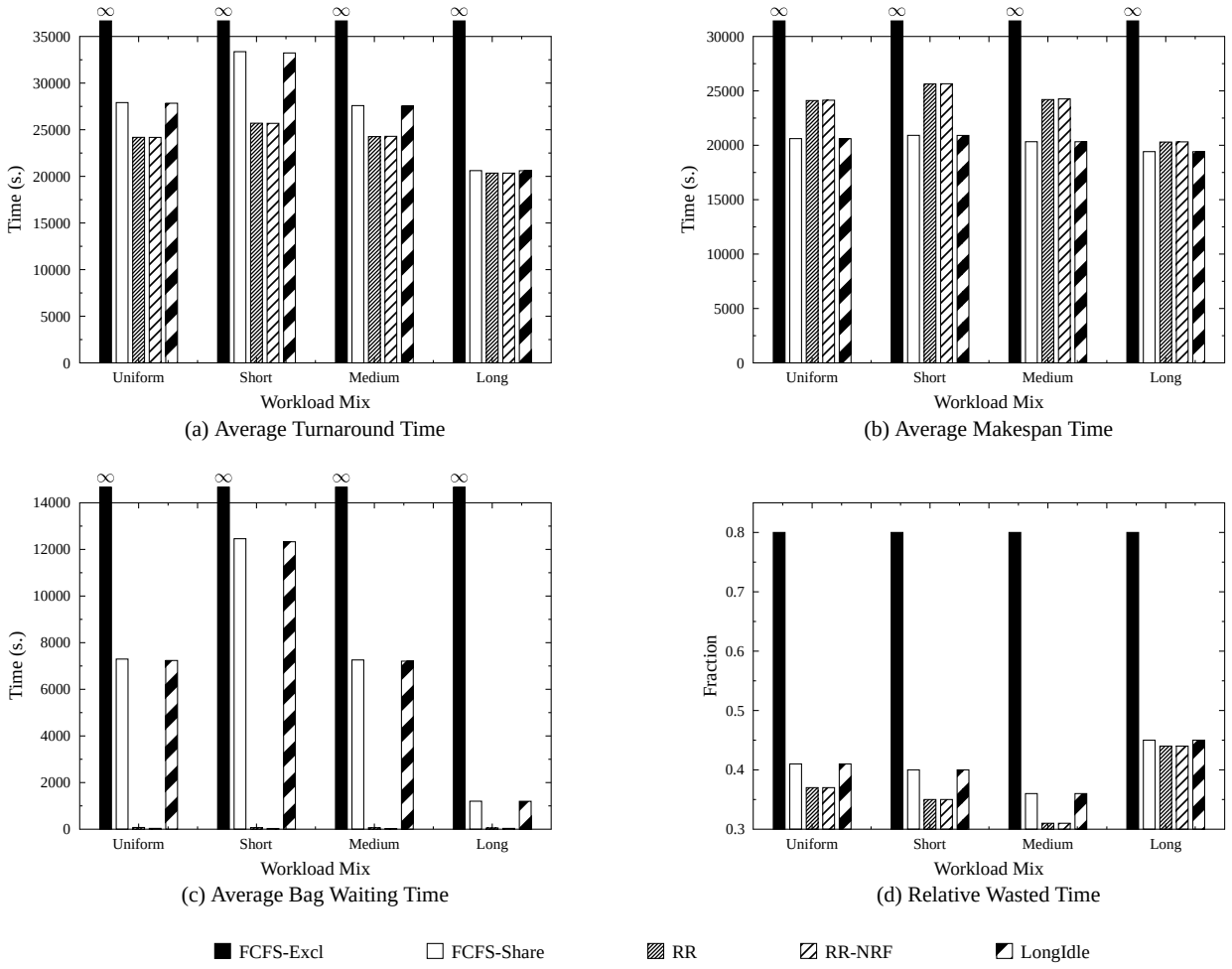

Fig. 10. Results for the High-Avail/Homogeneous Desktop Grid configuration and Multi-Class/Low-Intensity workloads

(Fig. 10) we note that, for all workload mixes, the turnaround time obtained for FCFS-Excl (Fig. 10(a)) grows to an unlimited value, as consequence of the simultaneous growth of the makespan (Fig. 10(b)) and of the waiting time (Fig 10(c)). This is due to the fact that FCFS-Excl allocates all the resources to a single BoT application – thus increasing the number of replicas per task – even in situations like these, where replication yields marginal benefits because of the homogeneity and high availability of machines. This effect is confirmed also by the results concerning the RWT (Fig. 10(d)), that indicate that the percentage of useless replicas generated by FCFS-Excl is about 80% for all the workload mixes.

For the other scheduling policies, we note that, RR and RR-NRF perform better (for the *Uniform*, *Short*, and *Medium* workload mixes) or slightly better (for the *Long* mix) than the other ones. The results shown in Fig. 10(b) indicate that this gap is not due to the average makespan (that is higher for RR and RR-NRF), but – as can be seen from Fig. 10(c) – it can be ascribed to the waiting time. The explanation of this phenomenon recalls that already given for single class workloads: since the average number of tasks in a BoT is comparable to (*Uniform*, *Short*, *Medium*) or much smaller than (*Long*) the number of machines of the Desktop Grid, there are enough resource to simultaneously schedule several BoTs (as RR

and RR-NRF do), thereby reducing their waiting time with respect to the other policies, and at the same time to allocate to each of them enough resources, thereby keeping their makespan within reasonable limits. Conversely, for the *Long* workload mix, we observe that all the scheduling policies but (as already discussed) FCFS-Excl exhibit similar performance. This is due to the fact that the *Long* workload mix is characterized by BoTs having an average number of tasks much smaller than the number of machines of the Desktop Grid (~ 0.5 tasks/machine). Therefore, in this case there are enough resources to concurrently run several BoTs, and there are so many resources for each BoT that even RR and RR-NRF creates a significant number of replicas per task (as can be deduced by the fact that their values of RWT are practically identical to that corresponding to FCFS-Share and LongIdle – see Fig. 10 (d)). Consequently, there are no appreciable differences in their performance.

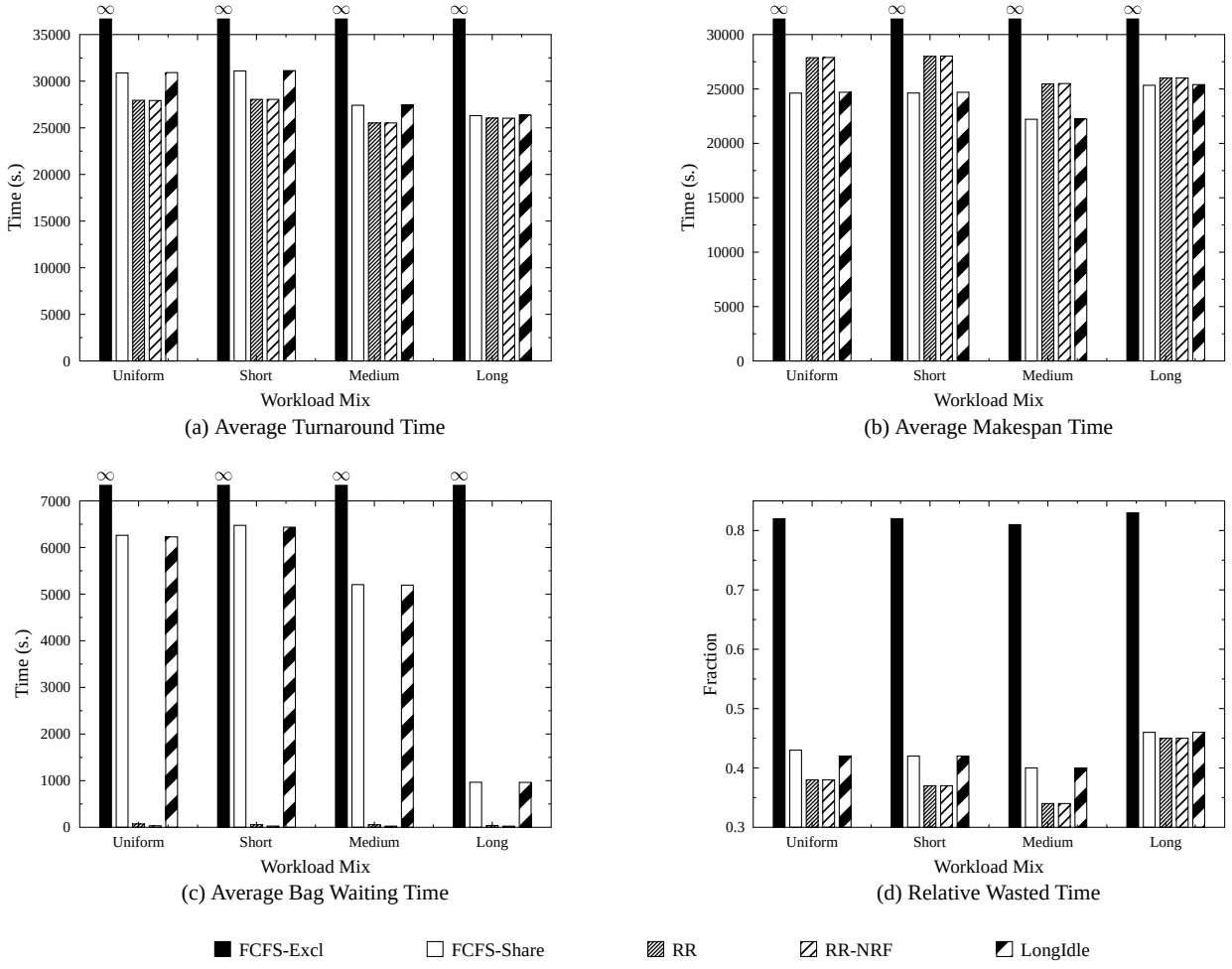

Fig. 11. Results for the High-Avail/Heterogeneous Desktop Grid configuration and Multi-Class/Low-Intensity workloads

When the machines of the Desktop Grid are heterogeneous (but still highly-available), the results for low-intensity (shown in Fig. 11), are similar to those obtained for the homogeneous configurations. As in the latter case, RR and RR-NRF perform better (*Uniform*, *Short*, *Medium*) and slightly better (*Long*) than the other strategies. The only notable difference with respect to the homogeneous case is that in this scenario the performance gap between the RR and RR-NRF and the other policies is smaller. This is due to the fact that when resource are heterogeneous, replication pays off, as it allows to better tolerate poor scheduling decisions due to the lack of resource information. As mentioned before, FCFS-Excl, FCFS-Share and LongIdle, being more prone to replication, exhibit performance better than in the homogeneous

case, while the opposite is true for RR and RR-NRF. However, despite this difference in performance, also in this case RR and RR-NRF are clear winners in terms of their ability of profitably exploit the available resources.

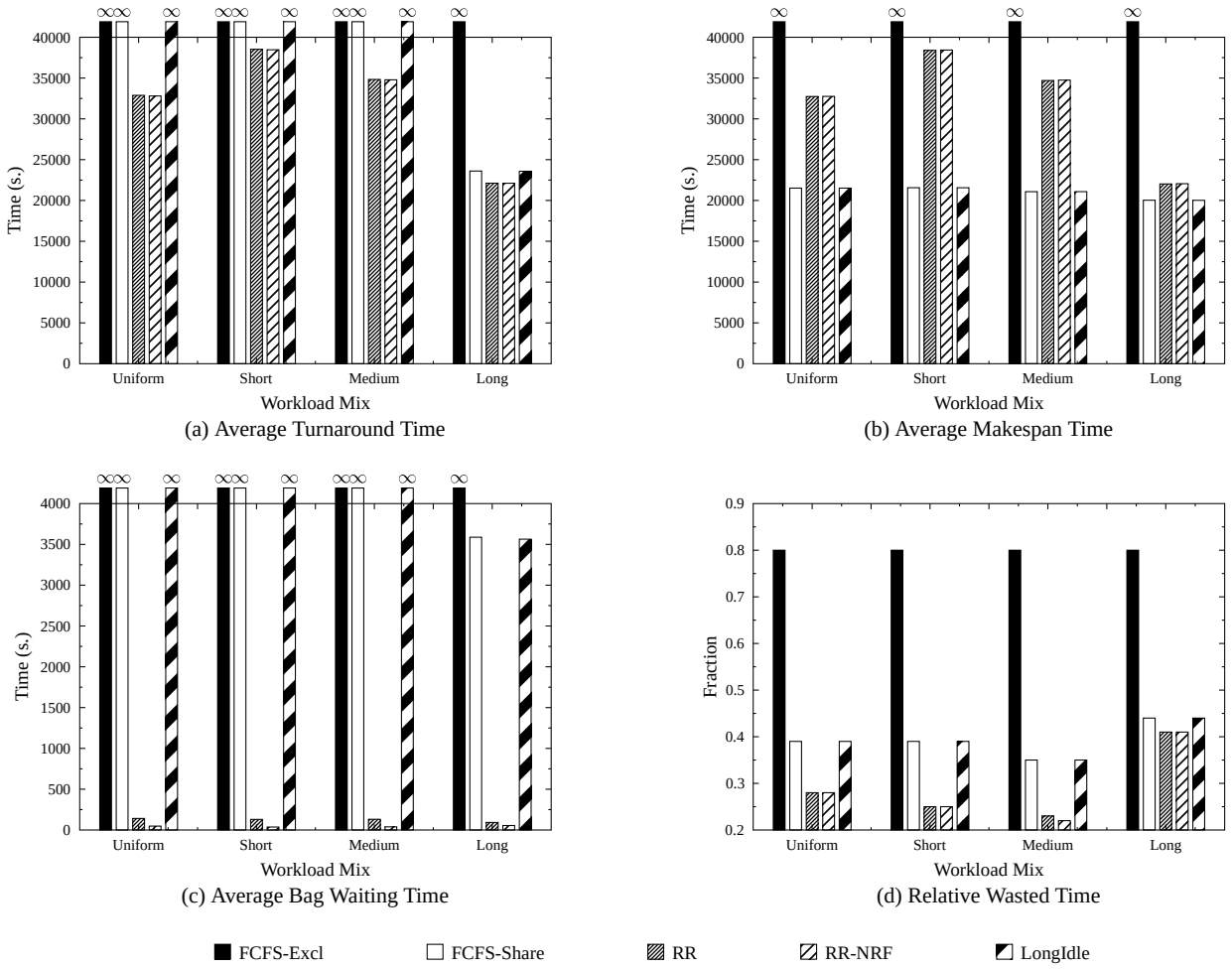

Fig. 12. Results for the High-Avail/Homogeneous Desktop Grid configuration and Multi-Class/High-Intensity workloads

The results obtained for Desktop Grid configurations when the workload intensity is high (see Figs. 12 and 13) tell a different story. In particular, FCFS-Excl, FCFS-Share, and LongIdle fail to yield a finite value for the turnaround time since, as can be seen from Figs. 12(c) and 13(c), their waiting time is practically unbounded, while this is not the case of RR and RR-NRF. Only for the *Long* workload mix, the FCFS-Share and LongIdle are able to yield a finite value for the turnaround time (albeit worse than that corresponding to RR and RR-NRF). This is due to the fact that, again, for the *Long* workload mix there are enough resources to simultaneously schedule several BoTs, thereby reducing their waiting time, and to give to each of them enough resources to minimize their makespan as well. Also in these cases, RR and RR-NRF are clear winners even in terms of RWT (see Figs. 12(d) and 13(d)).

When the Desktop Grid availability decreases to the medium value (corresponding to 75%) our results (see Figs. 14 through 17) show that, again, RR and RR-NRF perform better (*Uniform*, *Short*, *Medium*) and slightly better (*Long*) than the other strategies. The only notable difference with respect to the high availability case is that in this scenario – for the *Long* workload mix and low-intensity workload – the FCFS-Excl scheduling policy is able to yield a finite value for the average makespan time (see Fig. 14(b) and Fig. 15(b)). This is due to the fact that when resources are not reliable, replication pays off, as it

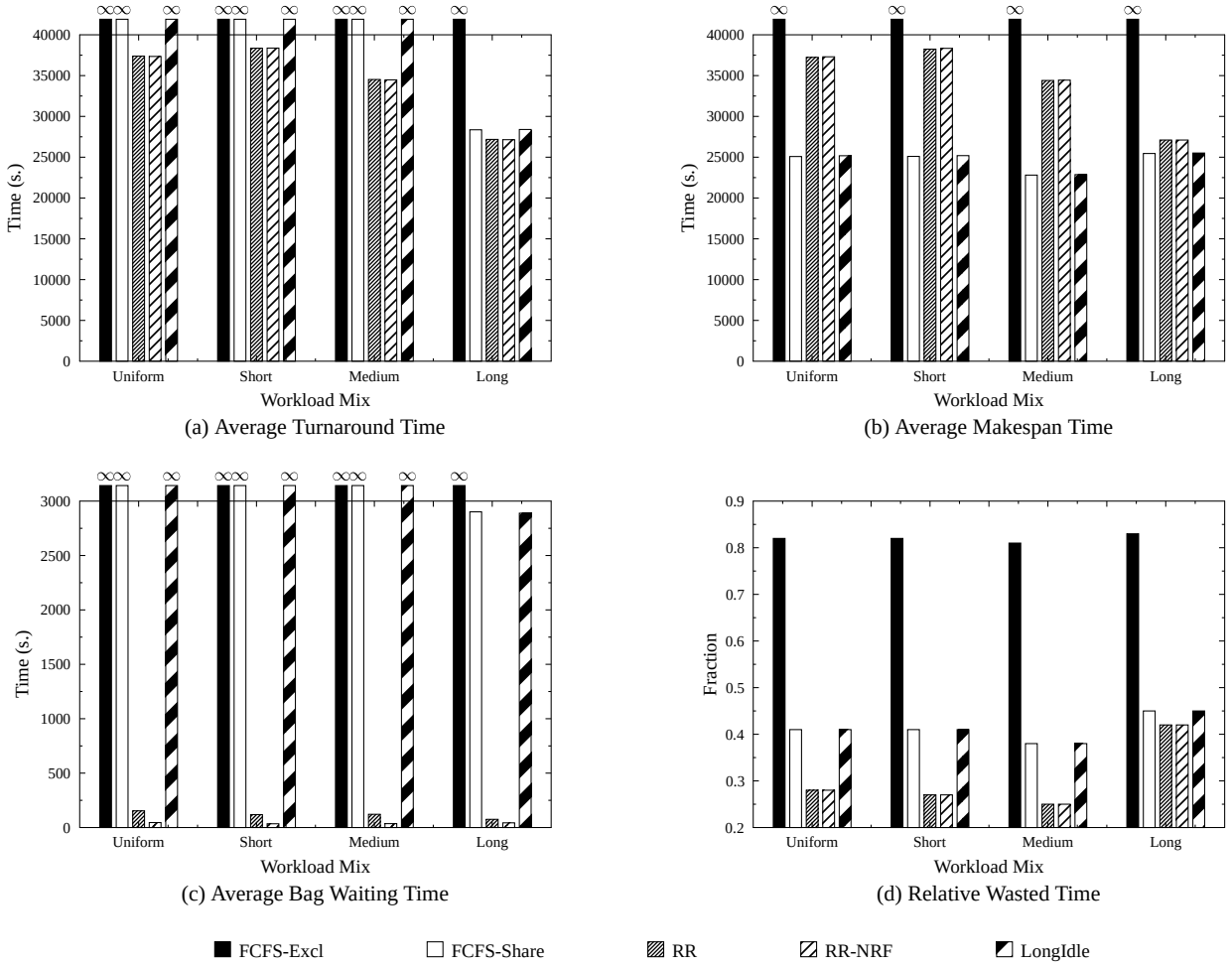

Fig. 13. Results for the High-Avail/Heterogeneous Desktop Grid configuration and Multi-Class/High-Intensity workloads

allows to better tolerate machine failures. However, despite this difference in performance, also in this case RR and RR-NRF are clear winners for scenarios characterized by resource with medium availability.

Finally, the results obtained for low availability configurations corresponding, as already mentioned, to Volunteer Desktop Grids (not reported here for the sake of brevity), show that for low intensity workloads, all the strategies behave exactly as observed for high and medium availability values. That is, RR and RR-NRF perform better (*Uniform*, *Short*, *Medium*) and slightly better (*Long*) than the other strategies. However, in this scenario, we observe that for high-intensity workloads all the scheduling policies always fail to yield a bounded turnaround time for the Short workload mix. This is not unexpected, since the large number of tasks for each bag and the high load factor make the system work under saturation, so there is no room to create enough replicas to tolerate poor scheduling decisions or machine failures.

3) Discussion: The results reported in the previous sections allow us to draw some general conclusions about the performance and the efficiency of the various scheduling policies considered in this paper.

As discussed in the previous sections, FCFS-Excl (that we recall is a direct extension of the scheduling algorithm used by state-of-the-art Desktop Grid platforms to schedule workloads composed by a single BoT a time) never yields performance better than all the other strategies. As a matter of fact, for many workloads and Desktop Grid configurations RR and RR-NRF perform better than all the other algorithms both in terms of performance and efficiency, while when the opposite is true, FCFS-Excl performs always

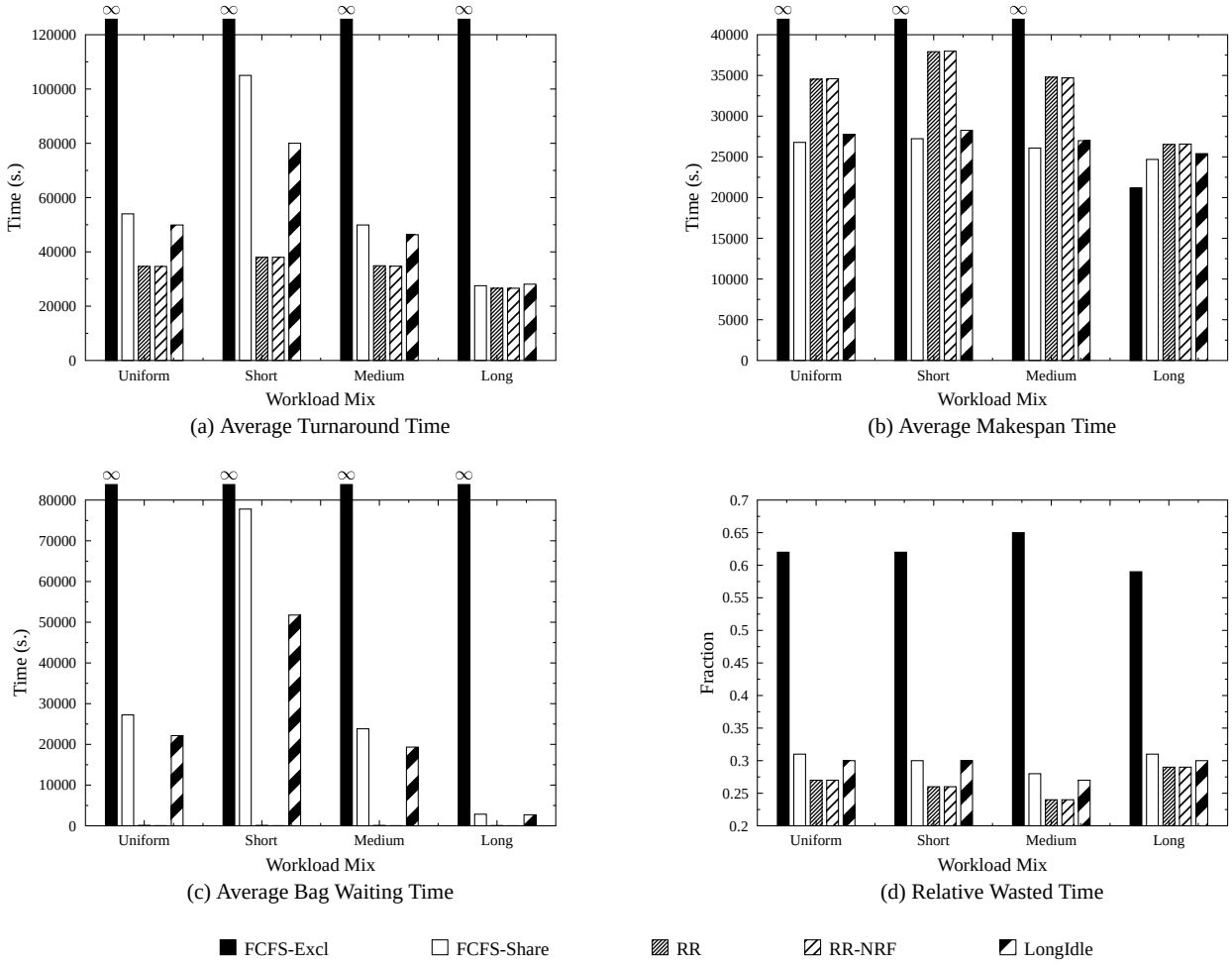

Fig. 14. Results for the Medium-Avail/Homogeneous Desktop Grid configuration and Multi-Class/Low-Intensity workloads

worse (or, in some cases, the same) than FCFS-Share both in terms of efficiency and performance. This demonstrates our initial claim that the vanilla FCFS scheduling algorithm cannot be used in situations where multiple BoT are simultaneously submitted for execution.

As a second consideration, we observe that for all the scenarios and workloads considered, FCFS-Share and LongIdle results in the same performance and efficiency levels. Therefore, being FCFS-Share easier to implement, it should be preferred to LongIdle in those situations where RR and RR-NRF perform worse. Analogously, RR and RR-NRF always exhibit the same performance and efficiency levels but, being RR easier to implement, it should be preferred over RR-NRF.

This said, let us provide an answer to the question whether any of these two scheduling policies must be preferred over the other one for all the Desktop Grid scenarios and application workloads we considered in this paper.

As it appears from the results concerning the amount of wasted time, RR is always more efficient than FCFS-Share for all the Desktop Grid configurations and application workloads we considered, that is it wastes less CPU cycles. This has the consequence that, given a performance goal that must be met (e.g., an upper threshold that must not be exceeded by the turnaround time), RR requires less resources than FCFS-Share to achieve it. Furthermore, for a fixed amount of computing power provided by the Desktop Grid, RR is able to better deal with sudden workload spikes than FCFS-Share, as it is always able to spare a higher amount of resource power thanks to its ability of wasting less computing capacity.

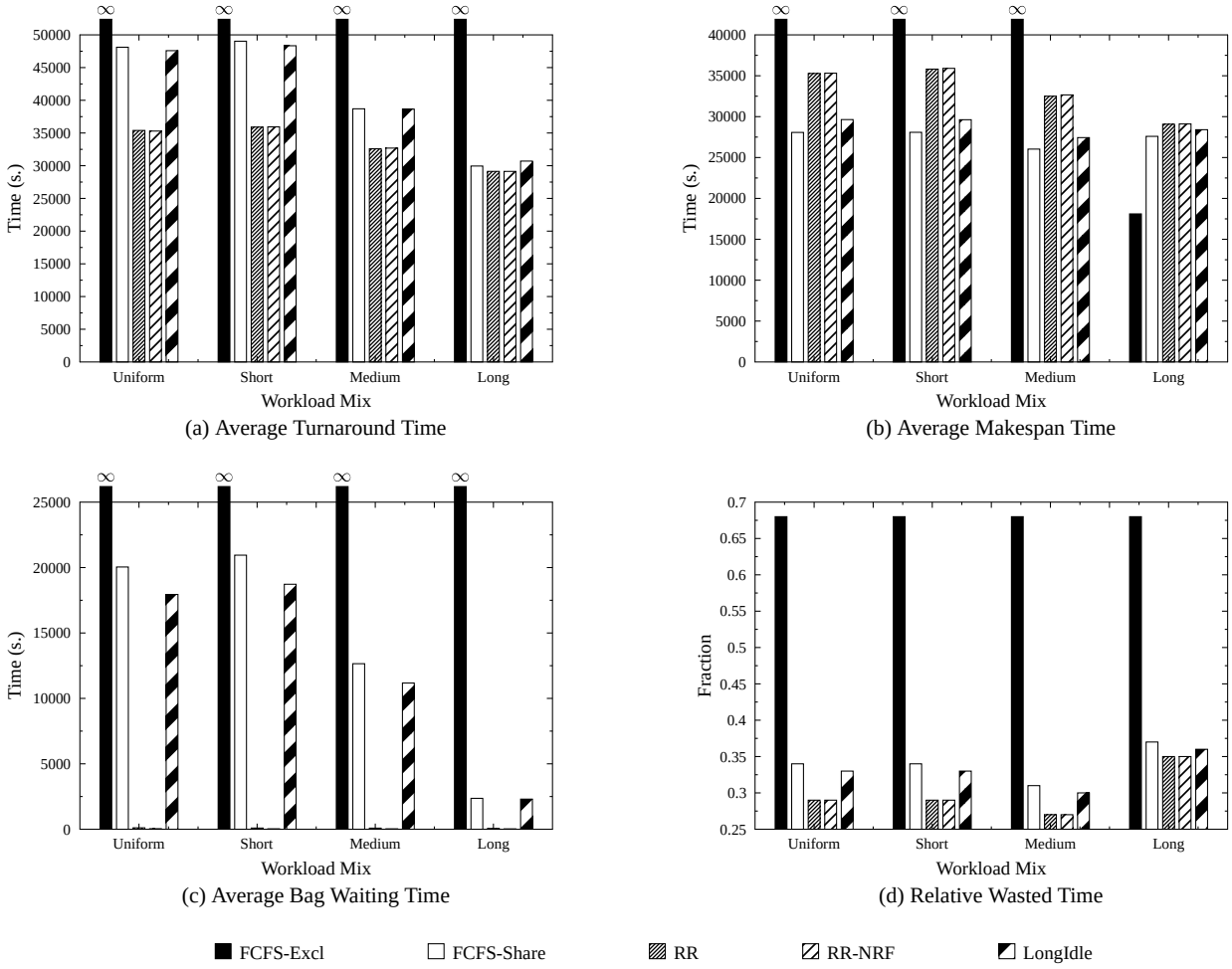

Fig. 15. Results for the Medium-Avail/Heterogeneous Desktop Grid configuration and Multi-Class/Low-Intensity workloads

The results concerning performance, however, indicate that no single scheduling policy outperforms the other ones for all Desktop Grid configurations and application workloads we considered, although RR performs better than FCFS-Share in a higher number of situations. The results concerning single class workloads, summarized in Table V, indicate indeed that FCFS-Share has to be preferred over RR only for workloads featuring very small and small task granularities (regardless of Desktop Grid configuration and workload intensity), while RR performs better in the other cases. The only exceptions are represented by the HighAvail/Homogeneous Desktop Grid configurations and the *All_M* workload mix, where FCFS-Share and RR perform the same (indicated by the $\sim$ symbol in the corresponding table entries), and by the Low-Avail/Heterogeneous configuration and *All_L* workload mix, where both policies fail to yield a finite value for the turnaround time.

The situation is instead different for multiple class workloads, whose results are summarized in Table VI, where we observe that RR always performs better than FCFS-Share. The only exception to this rule is represented by the Homogeneous Desktop Grid configurations and the Long workload mix, where both strategies exhibit very similar performance (denoted by the symbol $\sim$).

Therefore, we can conclude that, while in terms of efficiency RR should always be preferred over the other scheduling policies, for single class workloads consisting of fine grain tasks FCFS-Share represents the best choice, while for all the other situations RR provides higher performance advantages.

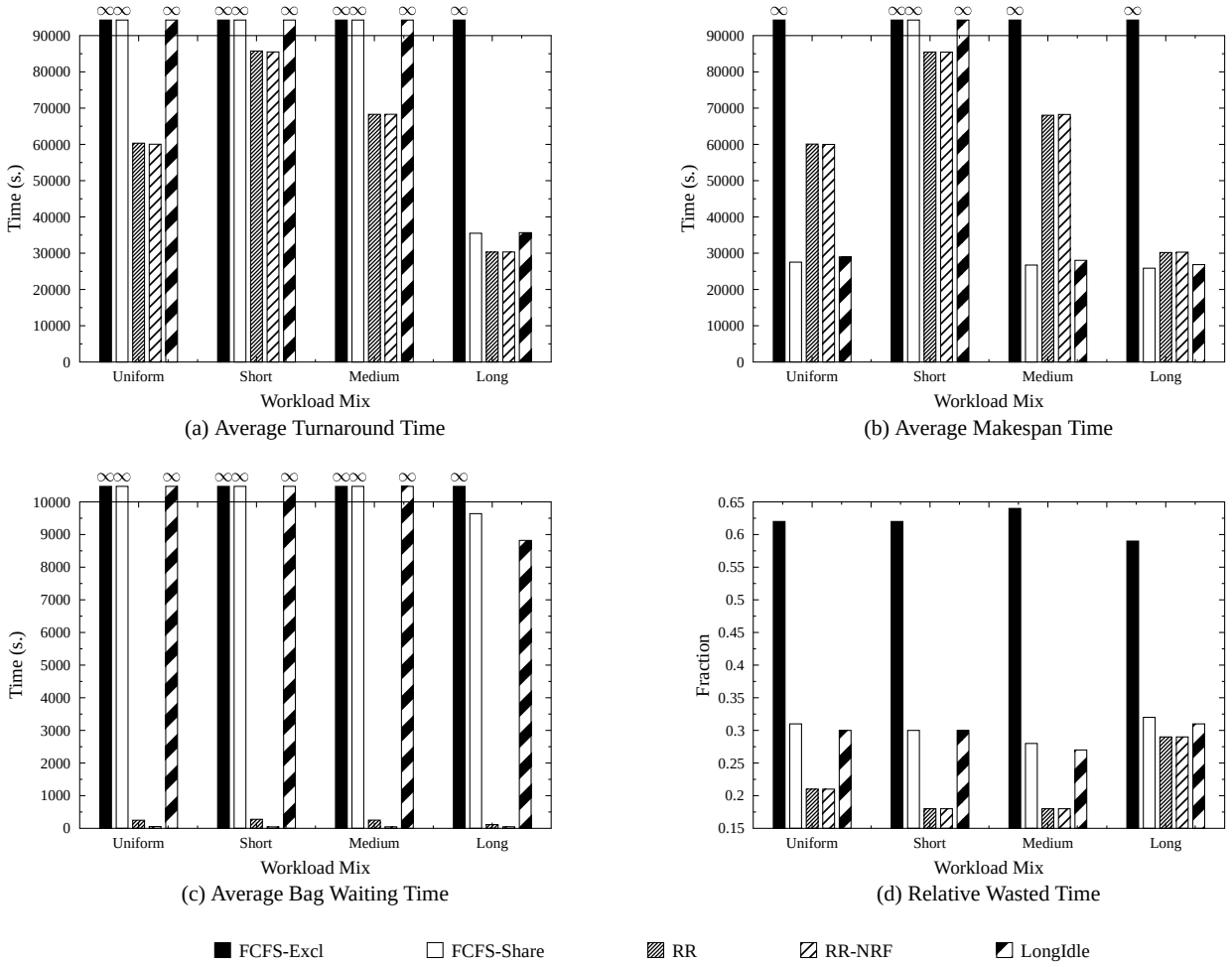

Fig. 16. Results for the Medium-Avail/Homogeneous Desktop Grid configuration and Multi-Class/High-Intensity workloads

TABLE V

COMPARISON OF THE TURNAROUND TIME FOR SINGLE CLASS WORKLOADS

Desktop Grid configuration		**Workload Mix**							
		All_VS		***All_S***		***All_M***		***All_L***	
		L_{low}	L_{hi}	L_{low}	L_{hi}	L_{low}	L_{hi}	L_{low}	L_{hi}
High-Avail	Homogeneous	FCFS-Share	FCFS-Share	FCFS-Share	FCFS-Share	$\sim$	RR	RR	RR
High-Avail	Heterogeneous	FCFS-Share	FCFS-Share	FCFS-Share	FCFS-Share	$\sim$	RR	RR	RR
Low-Avail	Homogeneous	FCFS-Share	FCFS-Share	FCFS-Share	FCFS-Share	FCFS-Share	RR	RR	∞
Low-Avail	Heterogeneous	FCFS-Share	FCFS-Share	FCFS-Share	FCFS-Share	RR	RR	RR	∞

TABLE VI

COMPARISON OF THE TURNAROUND TIME FOR MULTIPLE CLASS WORKLOADS

Desktop Grid configuration		**Workload Mix**							
		Uniform		**Short**		**Medium**		**Long**	
		L_{low}	L_{hi}	L_{low}	L_{hi}	L_{low}	L_{hi}	L_{low}	L_{hi}
High-Avail	Homogeneous	RR	RR	RR	RR	RR	RR	$\sim$	RR
High-Avail	Heterogeneous	RR	RR	RR	RR	RR	RR	$\sim$	RR
Med-Avail	Homogeneous	RR	RR	RR	RR	RR	RR	$\sim$	RR
Med-Avail	Heterogeneous	RR	RR	RR	RR	RR	RR	$\sim$	RR

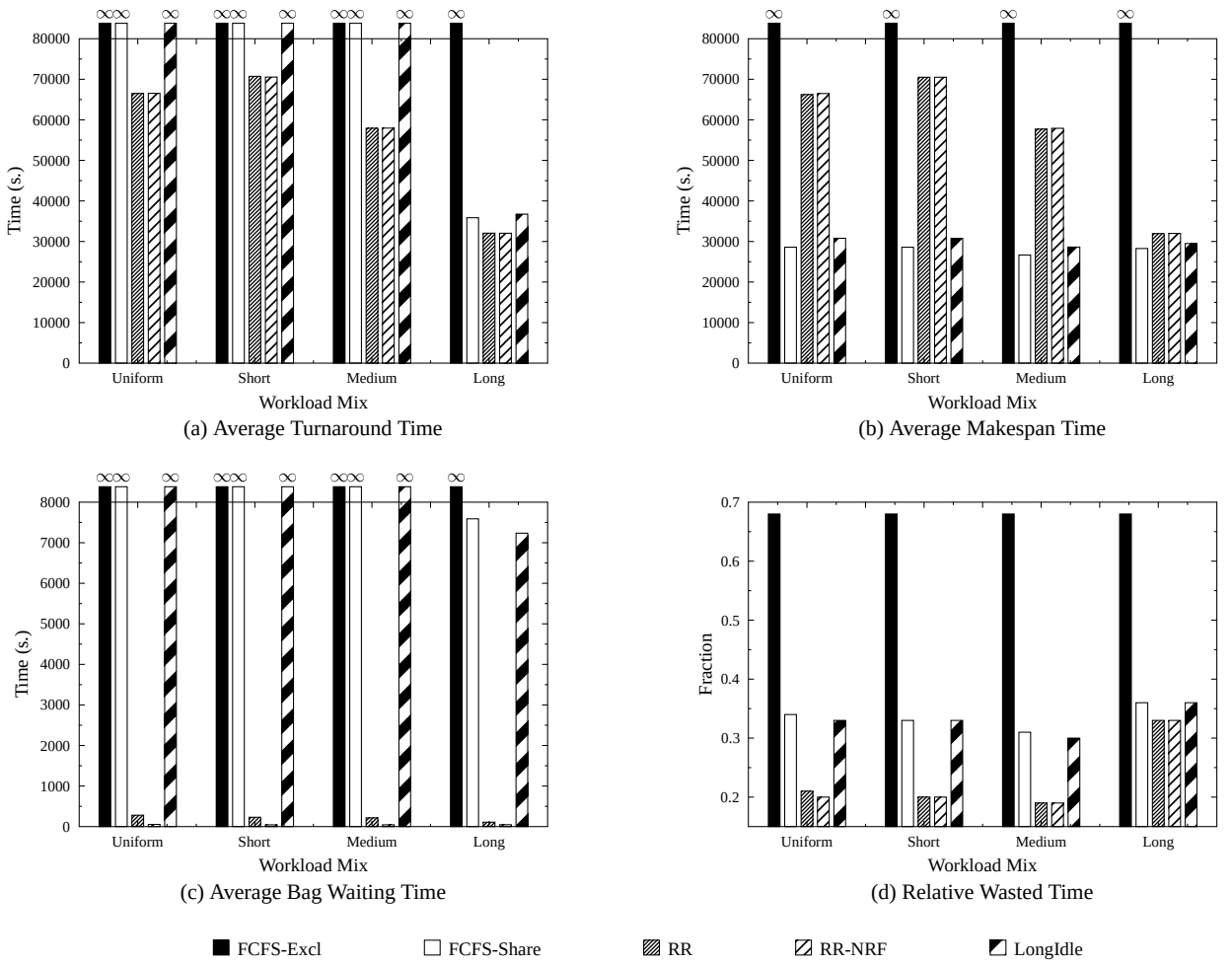

Fig. 17. Results for the Medium-Avail/Heterogeneous Desktop Grid configuration and Multi-Class/High-Intensity workloads

V. Conclusions and future work

Desktop Grid infrastructures are becoming commonplace not only in the academia, but also in the enterprise. Consequently, their employment as general-purpose computing platforms, exploited by different user communities running a variety of different BoT applications at the same time is continuously increasing. This motivates the need of scheduling algorithms able to simultaneously promote application performance and efficiently use available resources. The current state-of-the-art of scheduling algorithm for Desktop Grid offers solutions able to effectively schedule workloads where a single BoT is submitted (and, hence, needs to be scheduled) at a time but, as shown in this paper, these strategies fail to achieve this goal.

In this paper we have proposed a set of novel scheduling strategies specifically conceived to deal with scenarios where multiple BoT applications are simultaneously submitted to a Desktop Grid, that have been shown to provide an appropriate answer to the performance and efficiency needs characterizing modern Desktop Grid infrastructures. Our results clearly indicate that (a) these strategies enable applications to achieve performance better than those attainable by using vanilla FCFS schedulers, (b) simple Round Robin (RR) always results in better resource utilization, and in many cases in better performance too (especially for workloads featuring BoTs characterized by different task granularities), and (c) a straightforward variant of FCFS in which free resources that are not strictly necessary to the running BoT are allocated to the subsequent one greatly improves the performance of vanilla FCFS, and makes it suited to schedule single class workloads characterized by a small task granularity.

In addition to the above advantages, our strategies have the additional benefit of being knowledge-free, that is they neither require nor rely on any information concerning resources and applications. As such, they are easier to implement, deploy, and use on Desktop Grid infrastructures, whose relative resource instability makes very hard (if not impossible) the task of collecting accurate information about resources or applications.

The scheduling strategies we presented, albeit already providing a suitable answer to the scheduling needs of contemporary Desktop Grids infrastructures and workloads, still provide room for improvements. Among those, we mention the possibility of using individual BoT scheduling algorithms that use dynamic replication strategies (rather than the static one used in this paper), as this intuitively should further reduce the amount of wasted CPU cycles. Another possible avenue of research worth exploring is represented by the usage of knowledge-based scheduling algorithm for individual BoT (e.g., those proposed in [15]), in place of WQR-FT. This would make the scheduling strategy no more completely knowledge-free (the BoT selection policy, however, would still be knowledge-free), but might result in further application performance improvements. The adoption of a (partially) knowledge-based approach would also open a further avenue of research concerning the awareness of data placement. The scheduling algorithms presented in this paper are data placement agnostic, that is they do not consider the overhead due to data access, that may impact in a different way on the performance of different replicas of the same tasks executed on distinct machines. However, knowledge-free approaches cannot base their decisions on data placement information, as this would require the gathering of information concerning data access patterns for the various tasks, as well as those concerning data location. The migration towards a partial knowledge-based approach would allow us to combine the knowledge-free BoT selection policies presented in this paper with knowledge-based, data-aware resource selection policies, thus reducing the amount of information that must be provided to the scheduler with respect to fully knowledge-based strategies able to deal also with data placement.

References

[1] W. Cirne, F. Brasileiro, N. Andrade, L. Costa, A. Andrade, R. Novaes, , and M. Mowbray, "Labs of the world, unite!!!" *Journal of Grid Computing*, 2006.

[2] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, 2004, pp. 4–10.

[3] G. Fedak, C. Germain, V. Neri, and F. Cappello, "Xtremweb: A generic global computing system," in *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, 2001, p. 582.

[4] "United Devices Home Page," http://www.ud.com, visited on Sept. 5th, 2007.
[5] "Data Synapse Corp. Home Page," http://www.datasynapse.com, visited on Sept. 5th, 2007.
[6] S. Choi, H. Kim, E. Byun, M. Baik, S. Kim, C. Park, and C. Hwang, "Characterizing and Classifying Desktop Grid," in *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 743–748.
[7] D. Kondo, A. Chien, and H. Casanova, "Resource management for rapid application turnaround on enterprise desktop grids," in *Proc. of Supercomputing Conference*, 2004.
[8] "The great internet mersenne prime search," http://www. mersenne. org, 1997.
[9] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@home: an experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, 2002.
[10] "The Folding@home Project"," http://www.stanford.edu/group/pandegroup/folding, visited on Sept. 7th, 2007.
[11] "The FightAids@Home Project," http://fightaidsathome.scripps.edu, visited on Sept. 7th, 2007.
[12] W. Cirne and et al., "Grid computing for bag of tasks applications," in *Proc. of 3^{rd} IFIP Conf. on E-Commerce, E-Business and E-Government*, 2003.
[13] J. Smith and S. Srivastava, "A System for Fault-Tolerant Execution of Data and Compute Intensive Programs over a Network of Workstations," in *Proc. of EuroPar'96*, ser. Lecture Notes in Computer Science, vol. 1123, 1996.
[14] D. da Silva, W. Cirne, and F. Brasileiro, "Trading Cycles fro Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids," in *Proc. of EuroPar 2003*, ser. Lecture Notes in Computer Science, vol. 2790, 2003.
[15] C. Anglano, J. Brevik, M. Canonico, D. Nurmi, and R. Wolski, "Fault-aware scheduling for Bag-of-Tasks applications on Desktop Grids," in *Proc. of 7th IEEE/ACM International Conference on Grid Computing*. Barcelona, Spain: IEEE Press, Sept. 2006.
[16] C. Anglano and M. Canonico, "Fault-Tolerant Scheduling for Bag-of-Tasks Grid Applications," in *Proc. of the 2005 European Grid Conference*, ser. Lecture Notes in Computer Science, no. 3470. Amsterdam, The Netherlands: Springer, Berlin, Feb. 2005.
[17] D. Kondo, A. Chien, and H. Casanova, "Scheduling Task Parallel Applications for Rapid Application Turnardound on Enterprise Desktop Grids," *Journal of Grid Computing*, 2007, to appear.
[18] Y. C. Lee and A. Y. Zomaya, "Practical scheduling of bag-of-tasks applications on grids with dynamic resilience," *IEEE Trans. Comput.*, vol. 56, no. 6, pp. 815–825, 2007.
[19] D. Zhou and V. Lo, "Wave Scheduler: Scheduling for Faster Turnaround Time in Peer-Based Desktop Grid Systems," in *Proc. of 11th Workshop on Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, no. 3834. Boston, MA, USA: Springer, Berlin, June 2005.
[20] "The Compute Against Cancer Project," http://www.computeagainstcancer.org, visited on Sept. 7th, 2007.
[21] "The ShareGrid Project Home Page," http://dcs.di.unipmn.it, visited on Nov. 22nd, 2007.
[22] Y. Zhang, W. Sun, and Y. Inoguchi, "Predict task running time in grid environments based on CPU load predictions," *Future Generations Computer Systems*, vol. 24, pp. 489–497, 2008.
[23] M. Dobber, R. van der Mei, and G. Koole, "A prediction method for job runtime on shared processors: Survey, statistical analysis and new avenues," *Performance Evaluation*, vol. 64, pp. 755–781, 2007.
[24] W. Cirne, D. Paranhos, F. Brasileiro, and F. Goes, "On The Efficacy, Efficiency and Emergent Behavior of Task Replication in Large Distributed Systems," *Parallel Computing*, vol. 33, no. 3, pp. 213–234, April 2007.
[25] A. Chien, B. Calder, S. Elbert, and K. Bhatia, "Entropia: architecture and performance of an enterprise desktop grid system," *Journal of Parallel and Distributed Computing*, vol. 63, no. 5, pp. 597–610, 2003.
[26] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert, "Centralized versus Distributed Schedulers for Bag-of-Tasks Applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 5, May 2008.
[27] P. Domingues and J.G. Silva and L. Silva, "Sharing checkpoints to improve turnaround time in desktop grid computing," in *Proc. of the 20^{th} International Conference on Advanced Information Networking and Applications*. Vienna, Austria: IEEE Press, April 2006.
[28] P. Domingues and F. Araujo and L.M. Silva, "A DHT-based Infrastructure for Sharing Checkpoints in Desktop Grid Computing," in *Proc. of the 2^{nd} IEEE International Conference on e-Science and Grid Computing (e-Science '06)*. Amsterdam, The Netherlands: IEEE Press, December 2006.
[29] M. Bozyigit and M. Wasig, "User-level Process Checkpoint and Restore for Migration," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 2, pp. 86–96, Apr. 2001.
[30] M. Litzkow and T. Tannenbaum and J. Basney and M. Livny, "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System," Computer Sciences Department, University of Winsconsin, Tech. Rep. 1346, 1999.
[31] V.C. Zandy and B.P. Miller and M. Livny, "Process Hijacking," in *Proc. of 8^{th} Int. Symposium on High Performance Distributed Computing (HPDC-8)*. Redondo Beach, CA (USA): IEEE Press, Aug. 1999.
[32] S. Al Kiswany and M. Ripeanu and S.S. Vazhkudai and A. Gharaibeh, "stdchk: A Checkpoint Storage System for Desktop Grid Computing," in *Proc. of the 28^{th} Int. Conference on Distributed Computing Systems (ICDCS 2008)*. Beijing, China: IEEE Press, Jun. 2008.
[33] P. Domingues and F. Araujo and L. M. Silva, "A DHT-based Infrastructure for Sharing Checkpoints in Desktop Grid Computing," in *Proc. of the 2^{nd} IEEE Int. Conference on e-Science and Grid Computing*. Amsterdam, The Netherlands: IEEE Press, Dec. 2006.
[34] D. Nurmi, J. Brevik, and R. Wolski, "Modeling Machine Availability in Enterprise and Wide-area Distributed Computing Environments," in *Proc. of Euro-Par 2005*, ser. Lecture Notes in Computer Science, no. 3648. Lisboa, Portugal: Springer, September 2005.
[35] ——, "Automatic Methods for Predicting Machine Availability in Desktop Grid and Peer-to-Peer Systems," in *Proc. of 4^{th} Int. Workshop on Global and P2P Computing (GP2PC '04)*. Chicago, IL (USA): IEEE Press, April 2004.
[36] G. Harrison, "Stationary Single-Server Queuing Process with a Finite Number of Sources," *Operations Research*, vol. 7, no. 4, pp. 458–467, Jul.–Aug. 1954.
[37] L. Kleinrock, *Queueing Systems: Volume I – Theory*. New York, USA: Wiley Interscience, 1975.