# Ensemble Scheduling: Resource Co-Allocation on the Computational Grid

Jon B. Weissman and Pramod Srinivasan

Department of Computer Science and Engineering, University of Minnesota, Twin cities
jon@cs.umn.edu

**Abstract.** This paper investigates a novel scheduling paradigm, ensemble scheduling, that can be used to schedule mixed workloads consisting of single- and multi-resource applications in Grid environments. Ensemble scheduling exploits the pattern of prior resource requests for ensemble (multi-resource) applications to dynamically partition the workload across the system resources. We show via simulation that an ensemble-aware scheduler can provide better performance both for ensemble applications and the more common single resource applications in Grid environment.

## 1 Introduction

*Computational Grids* offer an attractive platform for resource-intensive parallel scientific applications. The Grid has the potential to enable applications that require resources contained in one or more remote sites. For example, applications may require coupling of multiple supercomputers for very large-scale simulations [1], access to remote instruments [5], or access to remote data sources [2], all spanning resources in multiple sites. The applications of interest in this paper require *temporal* coupling between the distributed resources because the application components interact directly during the course of the execution. We term the resources required by these applications as *resource ensembles* and the applications as *ensemble applications*. In contrast, embarrassingly parallel applications such as parameter studies and monte-carlo simulations while often scalable to multi-site resources, do not require simultaneous resource availability: resources can be freely added as the computation proceeds.

Grid infrastructure projects such as Legion [6] and Globus [4] are developing software that allow remote resources to be accessed and coupled together. However, ensuring that the resources required by applications are available simultaneously is a difficult problem. This problem has been termed resource co-allocation. Much of the research into this problem has focused on implementation mechanisms based on resource reservation. The primary issues are how to specify and guarantee a reservation for a single ensemble application across different sites with different administrative policies controlling resources. Increasingly, resource schedulers including the Maui scheduler and Portable Batch Scheduler (PBS) are planning reservation support.

It is clear that reservations and co-allocation support will become a standard part of production Grids, and ensemble applications will become more commonplace. We also believe that as Grids mature, they will play a role similar to today's production parallel supercomputers serving a variety of high end scientific applications including a **mixture** of traditional parallel application (e.g. single MPP-style applications) and ensemble applications. Production Grids will likely become rapidly over-subscribed much like production parallel supercomputers [3]. Consequently, efficient scheduling of Grid resources across this mixed workload becomes crucial to achieving high performance for **both** application classes. Prior work established that individual multi-resource application could be efficiently scheduled in Grid environments [9].

In this paper, we investigate how both types of applications can be efficiently scheduled together. We develop a new class of scheduling algorithms that exploit two specific properties of ensemble applications: (1) they often require specific resources (e.g. a particular telescope or a remote data source or several MPPs), and (2) they will repeat (e.g. the ensemble of resources requested by an application will repeat). The latter property is based on the view that ensemble applications will be run repeatedly over time. These properties suggest that ensemble applications will make predictable **and** repeatable requests to the system.

We compared the performance of *ensemble-aware* scheduling to simple scheduling policies such as random and shortest queue for a wide-range of simulated workloads. We found that it always outperforms random scheduling and that it outperforms shortest queue for a large number of characteristic workloads. In many instances, ensemble-aware scheduling leads to better performance for **both** single resource applications and ensemble applications. We present the application and scheduling models and a summary of our performance results.

## 2 Ensemble Scheduling

### 2.1 Application Model

*Ensemble* applications require resources in multiple sites. For simplicity, we assume that an ensemble application requires a single resource in one or more sites. In contrast, *regular* applications require resources in a single site only. We model the underlying computational Grid as a connected network of resources with dynamically varying available capacity. Resources are represented as having a type, capacity, and active queue, and a waiting queue (Fig. 1). The type indicates the class of the resource and its attributes. For example, `MPP-O2K` represents an Origin 2000 MPP. The capacity is the size of a resource in units that are meaningful to the type of resources. For example, it could be the number of processors within a MPP resource, the number of simultaneous connections allowed to a remote data server, etc. Applications request some number of resource units. As applications are allocated against the capacity, the available capacity *ac* decreases (perhaps to 0). The active queue is the set of applications that are currently using the resource (and how much they are using) and the waiting queue is the set of applications waiting to use some amount of the

resource. Applications are modeled completely by their resource requirements. Each resource is utilized by an application component running at the site where the resource is contained. Each application component communicates and synchronized periodically during the execution.

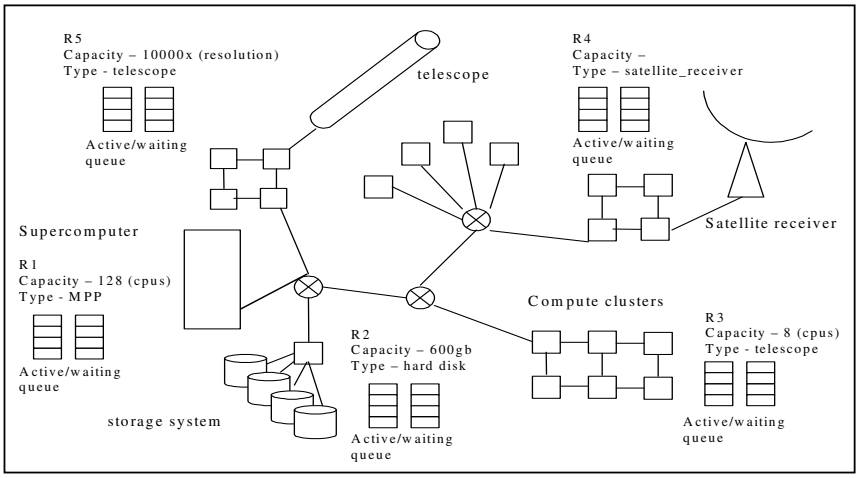


**Fig. 1.** Grid Model

For each desired resource, the application specifies the *min_amt* and *desired_amt* of the resource. The *min_amt* is the minimum amount required to run the application, and the *desired_amt* is the amount that if allocated, is predicted to be the best for the application. There is no benefit to allocating an amount of resources that exceeds the *desired_amt*. If the application were allocated its *desired_amt* for each requested resource, then it would attain its ideal execution time, *ideal_time*. This can be captured by an application-specific cost function *f* that determines execution time as a function of allocated resources, where $a_i$ is the allocated amount for resource $R_i$, for example,

$$f(a_1=desired_amt_1,\ a_2=desired_amt_2,\ \ldots) = ideal_time$$

The application also indicates the time frame that the resource is needed: $start_resource_time_i$ and $end_resource_time_i$ for each requested resource $R_i$ (offset from the application's beginning execution time). In this paper, we have adopted a co-allocation model for resources in which: *start_resource_time = 0* with the resource allocation held for the duration of the application (*end_resource >=_time ideal_time*). In production environments, an application may not always be able to obtain its *desired_amt*. If an allocated resource amount, $a_i$ < *desired_amti*, then the execution time must be penalized.

This captured by the function *f*. An example of *f* is the following:

$$f(\ldots) = max_i(desired_amt_i / a_i) * ideal_time$$

This particular *f* is constructed by assuming:

- All resources are held for the entire duration of the application
- Resource sharing has a linear penalty per resource

It applies to both regular applications (single resource) and ensemble applications (multiple resources). It proportionally penalizes the execution time by a linear term in the event that the desired amount is not available. Linear sharing models have been proposed to model a variety of resource types such as CPUs and networks. The scheduling model does not depend in any way on this formulation of *f*. Different penalty function can easily be used.

When an application is assigned to a resource and *min_amt* ≤ *ac*, the application is placed on the *active_queue* of the resource to start execution, and the available capacity *ac* of the resource is decremented by $a_i$. If an application is assigned to a resource, but its *min_amt* > *ac*, then the application must be queued on the *wait_queue* until additional resources are released by active applications upon their completion. Applications are allocated against the resource in a FCFS discipline. An ensemble application does not begin executing until **all** requested resources are available simultaneously with at least min_amt capacity. Applications enter the Grid with their resource requirements (Fig. 2).

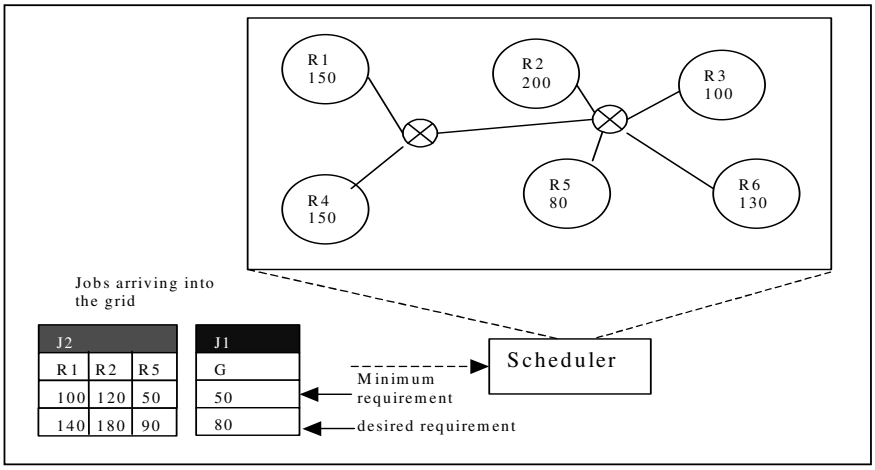


**Fig. 2.** Application Resource Requirements. $J_1$ is a regular application and $J_2$ is an ensemble application

## 2.2 Workload Model

The workload consists of a mixture of regular and ensemble applications. Regular applications are characterized by a single resource request and ensemble applications by multiple resource requests. Regular applications may request an amount of a specific resource (e.g. they must run on $R_1$) or a resource class (e.g. they must run on any `MPP-O2K`). Similarly, an ensemble application can request a specific resource or a resource class for each required resource. We believe that ensemble applications will have four key properties:

- The # of distinct resources they request will typically be small (e.g. 2 to 5)

- The # of distinct resources ensembles requested will also small
- Ensemble applications will be on the high-end of resource consumption
- Ensemble applications will repeat

The first property reflects current practice in which the number of distinct resources (e.g. MPPs) used within ensemble applications is small (<5) [7]. The second property reflects a belief that while ensemble applications will be more common in production Grids, they will still be in the minority as compared with single resource applications. The third property reflects a prediction that ensemble applications (particularly supercomputing coupling) are likely to be resource-intensive. For example, an application that wishes to use 3 MPPs probably requires close to all available processors within each MPP (otherwise a single MPP could be used). The first three properties are not crucial to scheduling algorithm, but they influence how the system workload is generated. The final property is critical to the scheduling algorithm and is based on our belief that in production Grids, ensemble applications will be run repeatedly. Because the effort in building ensemble applications will be large, this effort is justified for applications that are expected to be repeatedly. In fact, application repetition is also a property of many single resource MPP applications [3][8]. However, we believe it will be the rule, not the exception, for ensemble applications. Application repetition arises in several ways including: the application will be run on different data sets or requires interaction with remote devices or instruments that are providing periodic data.

Ensemble applications require co-allocation of all required resources. Two model of co-allocation are possible: *reservations* and *earliest-possible-time*. Reservations are needed if the application requires access to resources at very specific times (e.g. a telescope is providing data every 2 hours). Earliest-possible-time determines the earliest time that all desired resources will be free based on a prediction of the wait time for each resource. This information is known in our model since the time a resource will be held is simply the application execution time (which is provided by *f*). In this paper, we have experimented with *earliest-possible-time* with reservations left for future work. Ensemble scheduling is compatible with either co-allocation model.

Generating the application mix is controlled by a parameter which specifies the percentage of ensemble vs. regular applications. Ensemble applications are generated by following the four properties above. In particular, a small set of resource ensembles each with 2 to 5 resources is pre-generated (this satisfies properties 1 and 2). An ensemble application is generated by selecting one of these pre-generated ensembles with uniform probability (this satisfies property 4). The amount of desired resources for regular and ensemble applications are generated differently. Some analysis of supercomputing workloads indicated that jobs (i.e. regular applications) were much more likely to request a smaller number of processors (i.e. resources) [3]. A exponential pdf is used to model this relationship (Fig. 3, where the amount of requested resources is [1..MaxR]). Following property 3, we generated an inverted distribution for ensemble applications since we believe that they are more likely to request a resource amount near the full capacity of the resources.

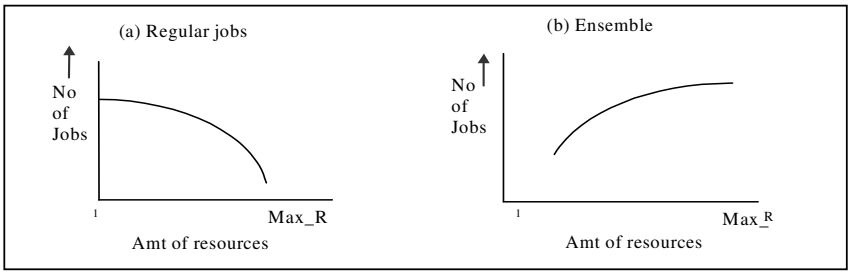


**Fig. 3.** Workload Generation. PDF is (a) is $4x^{1/2}$ and b) $1/4x^{1/2}$. Y-axis is the number of ensemble applications generated with a specific number of resources requested.

### 2.3 Scheduling Model

Regular and ensemble applications arrive to the system with their resource requirements. A scheduling algorithm is used to assign the incoming application stream to the available Grid resources. Applications are scheduled using a FCFS discipline on all resources. In this initial study, ensemble applications are generated to request a specific set of resources based on the belief that most ensemble applications are designed to access specific resources (e.g. $R_1$ and $R_2$). Therefore, scheduling them is straightforward. In future work, we plan to relax this model to allow certain application components within an ensemble application the freedom to specify a resource class (e.g. any `MPP-O2K`). In contrast, we assume that a fraction of regular applications may be freely run on any resource in the system. In this initial study, we set this fraction at 100%.

For a given synthetic workload containing a mix of regular and ensemble applications, we have compared three global scheduling approaches. Since ensemble applications use a fixed specific set of resources, they are simply queued on the resources they have requested. For regular applications, we have implemented the following algorithms:

*shortest_queue*
: pick resource with shortest waiting queue such that *min_amt >= max_capacity*

*random*
: pick a random resource such that *min_amt >= max_capacity*

*ensemble_aware*
: pick a resource that has participated in the fewest ensembles to date such that *min_amt >= max_capacity*

The first two algorithms require no explanation. The *ensemble_aware* algorithm tracks the resources that have been used to schedule prior ensemble applications in the past (when an ensemble application arrives to the system and is put on a set of resources, we increment a counter on each of those resources). The *ensemble_aware* algorithm will push regular applications away from resources used by ensemble applications. The intuition behind this algorithm is that by pushing regular application away from resources that ensemble application are likely to want in the future, we

should be able to improve the performance of ensemble application. In addition, because the components of ensemble applications will likely require a large amount of individual resources and will likely have a longer duration, it may also benefit regular application to avoid these resources. This is where we would expect *shortest_queue* to perform worse than *ensemble_aware* because *shortest_queue* cannot distinguish between the type of application component that is queued in on a resource. However, by restricting regular applications to execute on a subset of resources, this may also have a negative impact on regular applications. The question we seek to answer is: when can ensemble scheduling be expected to outperform traditional policies and vice-versa?

## 3 Results

Using simulation, we established the performance range of *ensemble_aware* scheduling as compared with traditional policies: *shortest_queue* and *random*. Ensemble and regular applications are generated with an exponential arrival distribution with requested resource amounts based on the exponential pdf of Fig. 3. The Grid is simulated to contain N resources, $R_1, \ldots, R_N$ and a capacity is computed for each using a uniform distribution [1 .. MaxR]. The application duration (i.e. the ideal execution time) for each application class is also taken as a uniform distribution. The duration of ensemble applications is scaled by a parameter (duration ratio) to capture their increased resource demand (from 1 to 20). The results establish that for a wide-range of parameter settings, *ensemble_aware* scheduling outperforms these traditional policies for regular applications (Fig. 4), particularly if the % of ensemble applications is small, between 5% and 20% (we expect this to be the more common case). For *shortest_queue*, as the % of ensemble applications increase, the performance becomes similar. The reason is that as the % of ensemble applications increases, the queue length on ensemble resources begins to grow, and *shortest_queue* will naturally begin to avoid these resources. As the duration of ensemble applications approaches that of regular applications, the performance of *ensemble_aware* scheduling falls off. If the durations are similar, then regular application could run on ensemble resources with little additional penalty, but stacking them on a subset of non-ensemble resources causes ensemble scheduling performance to worsen.

However, *ensemble_aware* scheduling is always better for ensemble applications. As the % of resources that can be used for ensemble applications increases, the performance benefit of *ensemble_aware* scheduling increases (Fig. 5). However, *ensemble_aware* scheduling performs best when the duration of ensemble applications is similar to that of regular applications (Fig. 6). The reason is that as the duration of ensemble applications becomes much greater than regular applications, putting regular applications on ensemble resources does not impact the ensemble applications as much. Random scheduling also proved to be considerable worse than either *shortest_queue* or *ensemble_aware* (Fig. 7) and is plotted separately (note the change in y-scale). Collectively, these results suggest that a hybrid policy that is sensitive to application workload may have merit: when the duration of ensemble applications is similar to regular applications, use *shortest_queue*, but when the

duration of ensemble applications is a factor of 10 or more greater, then use *ensemble_aware* scheduling.

None of the scheduling policies presented exploit or require execution information about the application (this was used for simulation purposes only). A final question we considered was whether the use of information such as in a backfilling scheduler would improve performance for regular jobs? Since regular applications could sit behind longer running ensemble applications, they would seem to be perfect candidates for backfilling. We allowed regular applications to be placed on a resource and backfilled if they did not delay the applications already on the queue. The results indicate that backfilling can significantly boost performance for regular applications under either scheduling policy particularly as the % of ensemble applications increases (Fig. 8).

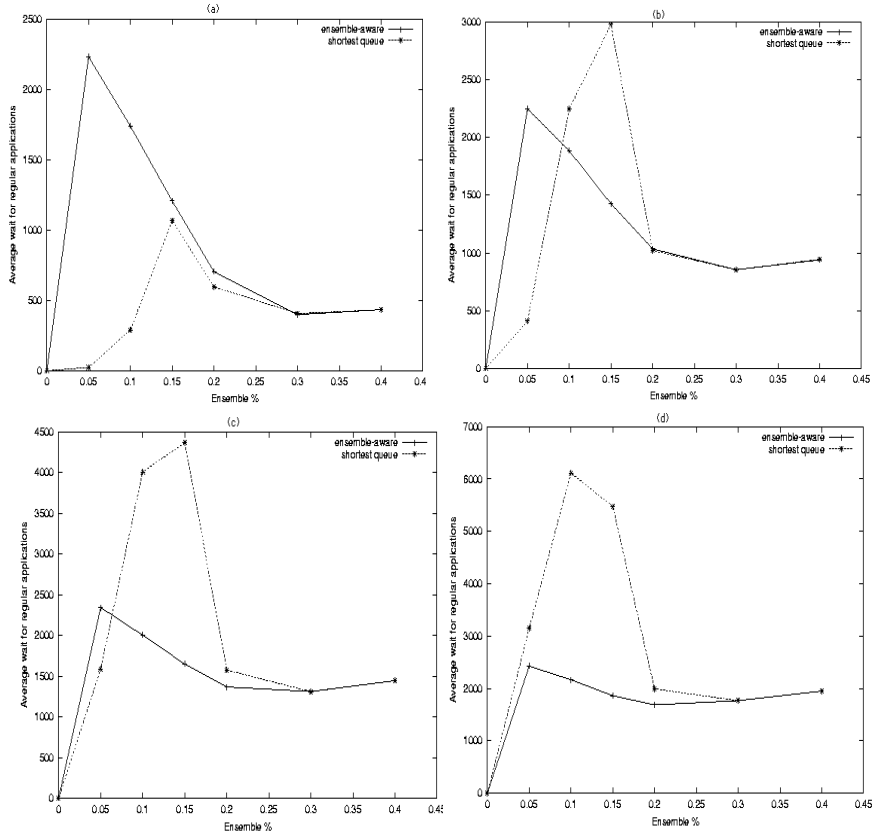


**Fig. 4.** Performance of Regular Applications. Duration ratio of ensemble to regular applications is (a) 5, (b) 10, (c) 15, and (d) 20. Duration ratios between the largest applications relative to the average size application were between 6 and 20 on the real workloads we examined [3].

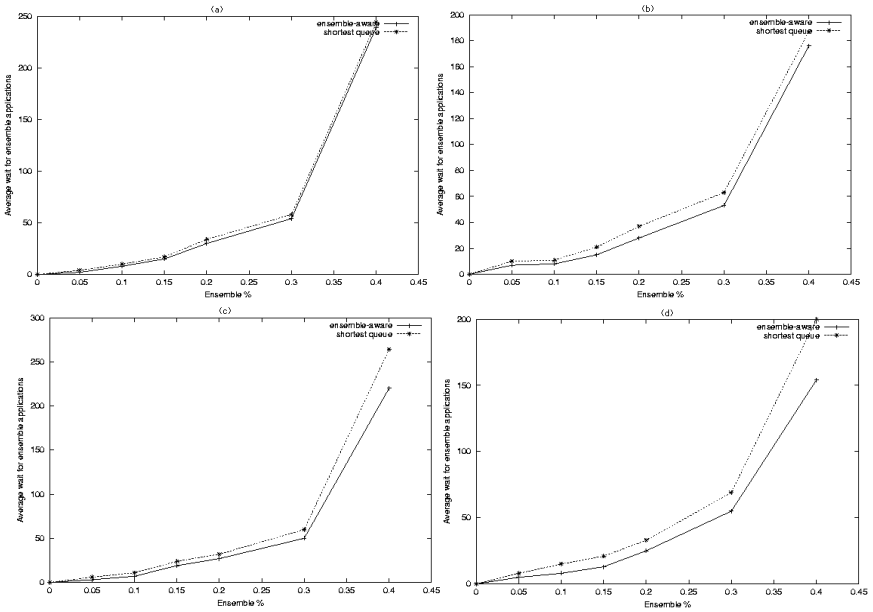


**Fig. 5.** Performance of Ensemble Applications. Percentage of resource available to ensemble applications is (a) .4, (b) .5, (c) .7, and (d) .9

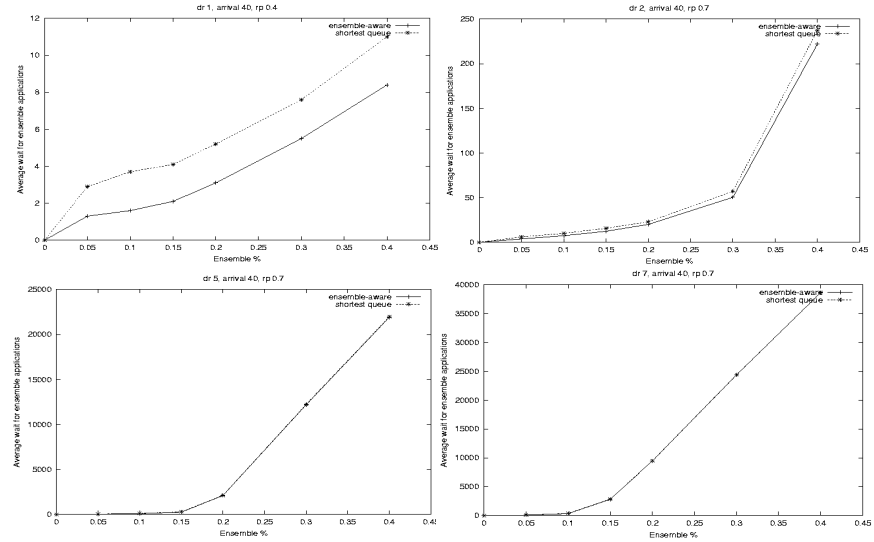


**Fig. 6.** Sensitivity of Ensemble Applications Performance to Duration Ratio. Duration ratio is 1, 2, 5, and 7, % of resource availability to ensemble applications is 0.7

.

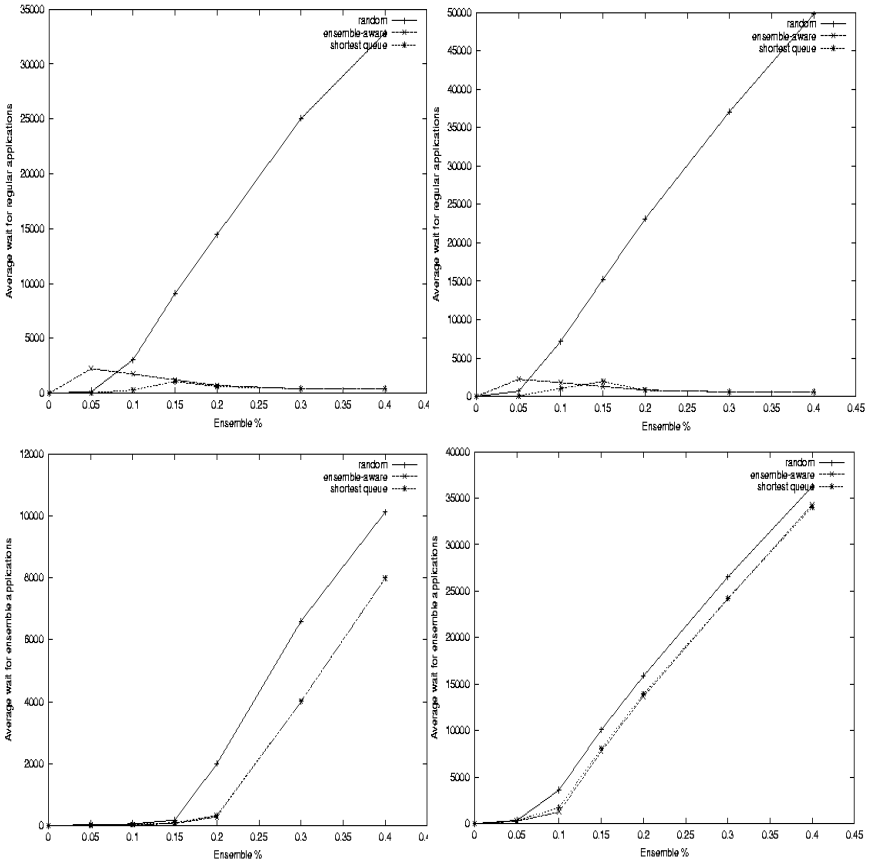


**Fig. 7.** Performance of Random Scheduling. Top two graphs are for regular applications, bottom two graphs for ensemble applications. Impact on regular applications is greatest. In all cases, random scheduling is worst.

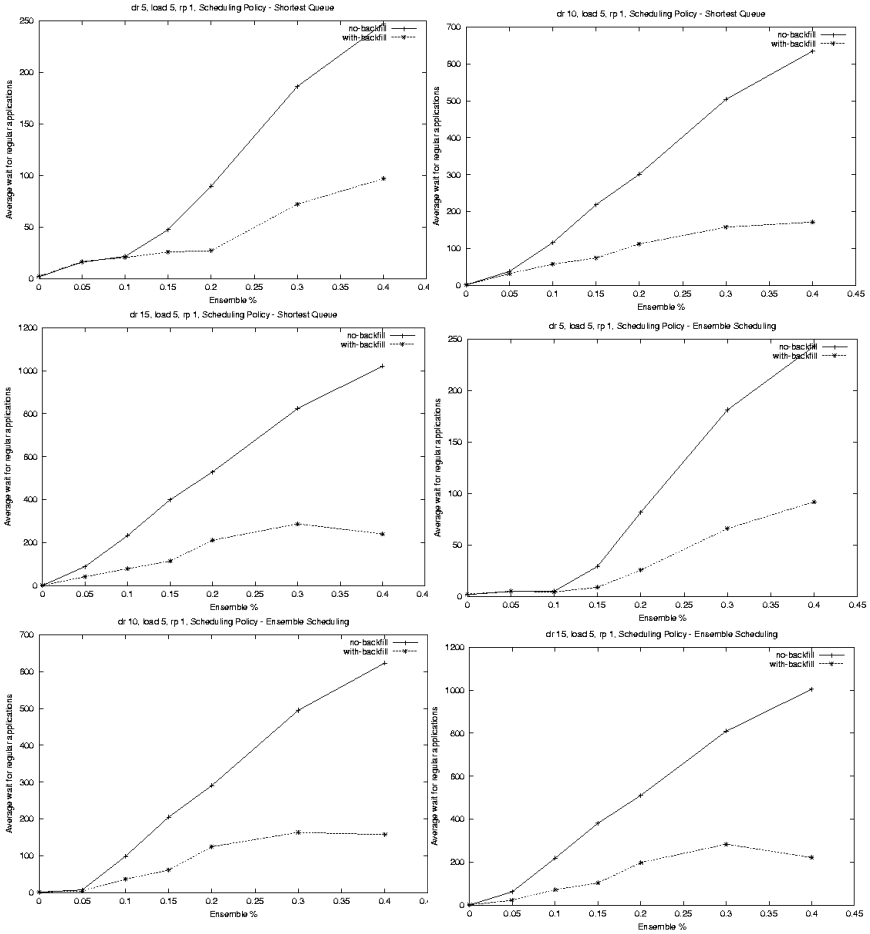


**Fig. 8.** Backfilling Regular Applications. Top graphs are for shortest queue and bottom are for ensemble_aware scheduling policies. Duration ratio is 5, 10, and 15 for each

# 4 Summary

We have presented a novel scheduling technique for scheduling mixed workloads consisting of single- and multi-resource ensemble applications in Grid environments. Ensemble-aware scheduling was shown to promote better performance for both regular and ensemble applications over a wide parameter range. In the vast majority of cases, it performed better or equal to *shortest_queue*. However, when the durations were similar, *shortest_queue* outperformed *ensemble_aware* scheduling for regular applications since the latter restricted the set of available resources to use. This

suggests that a hybrid scheduling policy that could adaptively switch between the two policies could be useful. Similarly, such a policy could be configured to bias towards either regular or ensemble applications. We also determined that the use of backfilling improved performance for regular applications under both scheduling policies. Future work includes applying backfilling to ensemble applications as well. Finally, our scheduling model considered only the earliest-possible-time co-allocation strategy. Incorporating reservations into our model and analyzing the resulting performance of the scheduling models is the subject of future work.

## 5 Bibliography


1. S. Brunett et al., "Implementing Distributed Synthetic Forces Simulations in Metacomputing Environments", *Proceedings of HCW* '98.
2. M. Faerman et al., "Adaptive Performance Prediction for Distributed Data-Intensive Applications", *Proceedings of SC99*, November 1999.
3. D. Feitelson, Parallel Workload Archive, http://www.cs.huji.ac.il/labs/parallel/workload/logs.htm, 1999.
4. I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", *international Journal of Supercomputing Applications*, 11(2), 1997.
5. *The Grid Blueprint for a New Computing Infrastructure,* (I. Foster and C. Kesselman, editors), Morgan Kaufmann Publishers, 1998.
6. A.S. Grimshaw and W.A. Wulf, "The Legion Vision of a Worldwide Virtual Computer", *Communications of the ACM,* Vol. 40(1), 1997.
7. P. Messina, "Distributed Supercomputing Applications", in *The Grid Blueprint for a New Computing Infrastructure,* (I. Foster and C. Kesselman, editors), Morgan Kaufmann Publishers, 1998.
8. W. Smith, I. Foster, and V. Taylor, "Predicting Application Run Times Using Historical Information", in *Job Scheduling Strategies for Parallel Processors*, D. Feitelson and L. Rudolph (eds.), Springer-Verlag, 1998, Lect. Notes Computer Science, vol. 1459.
9. J. Weissman, "Scheduling Multi-Component Applications in Heterogeneous Wide-Area Networks", *Proceedings of HCW* '99.