Journal Title
XX(X):1–15
©The Author(s) 2018
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

On-the-fly scheduling vs. reservation-based scheduling for unpredictable workflows

Ana Gainaru[1], Hongyang Sun[1], Guillaume Aupy[2], Yuankai Huo[1], Bennett A. Landman[1] and Padma Raghavan[1]

Abstract

Scientific insights in the coming decade will clearly depend on the effective processing of large datasets generated by dynamic heterogeneous applications typical of workflows in large data centers or of emerging fields like neuroscience. In this paper, we show how these big data workflows have a unique set of characteristics that pose challenges for leveraging HPC methodologies, particularly in scheduling. Our findings indicate that execution times for these workflows are highly unpredictable and are not correlated with the size of the dataset involved or the precise functions used in the analysis. We characterize this inherent variability and sketch the need for new scheduling approaches by quantifying significant gaps in achievable performance. Through simulations, we show how on-the-fly scheduling approaches can deliver benefits in both system-level and user-level performance measures. On average, we find improvements of up to 35% in system utilization and up to 45% in average stretch of the applications, illustrating the potential of increasing performance through new scheduling approaches.

Keywords

On-the-fly scheduling, reservation-based scheduling, neuroscience applications, unpredictable workloads

1 Introduction

High performance computing (HPC) has continued to advance infrastructure with multi-core heterogeneous nodes connected by fast networks with extensive storage capabilities in order to enable computational, modeling and simulation workflows to leverage massive levels of parallel processing. Such computational modeling workflows have predictable resource requirements, such as processing, storage, etc., which are utilized by the runtime system to deliver high performance.

Unlike traditional HPC workflows, in the last several years, new scientific fields are emerging that develop modeling and simulation workflows with unpredictable resource requirements. One such field is neuroscience whose workflows are broadly representative to this trend. In this paper, we characterize emerging neuroscience workflows that exhibit unpredictable resource requirements to inform the challenges and opportunities they pose for HPC including in particular scheduling.

Traditional scientific applications are focused on performance and have successfully utilized the power of HPC infrastructures. They include large monolithic applications having hundreds of thousands lines of code that have been developed by the community for years and that have been tunned to scale on HPC systems. For example, the NAMD, a widely used scientific application that implements a parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems, contains over 200K lines of code and has been developed and tunned by the community to achieve high performance on HPC architectures for more than 20 years. In contrast, neuroscience workflows focus around human subject studies and put a higher emphasis on productivity than performance. The many stages that compose a workflow have complex and dynamic dependencies between them. Stages are usually based on scripts and can change over time depending on the needs of each analysis as well as due to new functionalities that are being developed in order to gain scientific insight.

Reservation-based batch scheduling using priority queues and backfilling algorithms is the current de facto solution in implementing HPC schedulers. These systems are designed for traditional scientific applications and can have suboptimal performance for unpredictable workflows. There is currently a mismatch between the characteristics of these workflows and how their characteristics could be leveraged at runtime to deliver high performance unless all limitations are better understood. This paper highlights several essential differences between typical HPC scientific applications and unpredictable workflows by providing an in-depth analysis of the variability and unpredictability of the latter. Computational resources at massive scale need to be engaged in novel ways in order to accommodate the specific needs. We show that, by introducing “on-the-fly” scheduling into the runtime system, these workflows can leverage the HPC resources to achieve improved performance at both

[1] Vanderbilt University, Nashville, TN, USA
[2] Inria & Labri, Univ. of Bordeaux, Talence, France

Corresponding author:
Ana Gainaru, Vanderbilt University
Email: ana.gainaru@vanderbilt.edu

system and user levels, leading to better resource utilization and faster scientific discoveries.

The main contributions of this paper are the following:

- A detailed characterization of the resource requirements of neuroscience workflows highlighting what makes them fundamentally different compared to traditional HPC workloads. While we focus on neuroscience, we believe the resource variability and unpredictability of these workloads are characteristic of applications in other emerging fields.
- An extensive set of simulation results that evaluate current reservation-based batch schedulers for neuroscience workflows on both system-level and user-level performance measures (e.g., system utilization and average application stretch). We show that "on-the-fly" approaches that do not use reservations can be used successfully to deliver higher performance on these highly variable and unpredictable workloads.
- An illustration of how "on-the-fly" scheduling can be integrated into the current HPC runtime systems to form hybrid schedulers with a focus on two possible directions: reservation-free scheduling with priority queues and scheduling based on ranges of resource requirements.

The rest of this paper is organized as follows. Section 2 presents an overview of HPC runtime systems together with current theoretical and practical scheduling strategies. Section 3 provides a detailed analysis of neuroscience workflows highlighting characteristics typical of unpredictable workloads and key differences from traditional HPC applications. We show that these workflows depend on intricate features within input data and classical complexity functions can no longer be used to predict their resource requirements. Section 4 presents the simulation methodologies and describes the typical batch scheduler used by HPC systems together with "on-the-fly" schedulers. This section introduces the generation of synthetic workloads that mimic the characteristics of neuroscience workflows, the metrics to evaluate the performance, along with the simulation results of the schedulers. The results highlight the limitations of current HPC schedulers and demonstrate that unlike traditional batch schedulers, "on-the-fly" scheduling is able to more readily leverage the HPC resources to deliver high performance at both system and user levels. Section 5 discusses potentially interesting directions to accelerate these emerging fields either by including application-level optimizations or by more complex features that could be implemented in HPC schedulers. Finally, Section 6 provides brief concluding remarks.

2 Background and related work

This section presents an overview of the current scheduling solutions in HPC runtime systems from both theoretical (Section 2.1) and practical (Section 2.2) perspectives. We also highlight the challenge in scheduling for workloads with unpredictable resource requirements (Section 2.3).

2.1 Theoretical work on HPC scheduling

Many theoretical problems of scheduling a batch of jobs on a multiprocessor system have been shown to be NP-complete (Garey and Johnson (1990)). Thus, much research has been devoted to the design of approximation algorithms and heuristic solutions. In the seminal work, Graham (1966) showed that *list scheduling*, which arbitrarily orders the jobs in a list and schedules them one by one onto the least-loaded processor, is $(2 - \frac{1}{m})$-approximation with respect to the overall completion time of the jobs (a.k.a. makespan), where m denotes the total number of processors that are assumed to be identical. This result shows that list scheduling essentially produces a makespan that is guaranteed to be no longer than twice of the optimal solution. He also showed that the *longest job first (LJF)* heuristic, which orders the jobs in descending order of their execution times prior to applying list scheduling, achieves $(\frac{4}{3} - \frac{1}{3m})$-approximation for makespan. When the objective is to minimize the sum of response times of all jobs (a.k.a. total response time), the problem turns out to be solvable in $O(n \log n)$ time by the *shortest job first (SJF)* algorithm, which applies list scheduling while ordering the jobs in ascending order according to their execution times (Conway et al. (1967)). Similar performance guarantees have also been obtained under various other scheduling models, e.g., on non-identical processors, with job release times, allowing preemptions. The books by Pinedo (2008) and Brucker (2001) provide comprehensive summary of these classical results.

While both LJF and SJF heuristics assume that the job execution times are known a prior, it is not always true in practice. To model the unknown execution times, many papers have considered the *online non-clairvoyant* scheduling model, which assumes no prior knowledge about a job's execution time until it successfully completes on a processor. Since list scheduling is also applicable in this scenario, it achieves a makespan that is $(2 - \frac{1}{m})$-competitive against an optimal offline scheduler. To minimize the total response time, Motwani et al. (1993) showed that the *round robin (RR)* algorithm, which at any time ensures that all active jobs receive the same amount of processing time, is $(2 - \frac{2m}{n+m})$-competitive for scheduling a set of n batched jobs. More sophisticated non-clairvoyant algorithms have also been proposed (see, e.g., Shmoys et al. (1991); Chekuri et al. (1997); Becchetti and Leonardi (2004)) with proven competitive ratios under various objectives and scheduling assumptions. We refer to Pruhs et al. (2004) for a survey of results in this direction.

Another line of research to model the execution time uncertainty is that of stochastic scheduling. Most work in this scheduling paradigm assumes that the execution time of a job follows a known probability distribution. In this context, the *shortest and longest expected processing times first (SEPT and LEPT)* algorithms, which are stochastic variants of the SJF and LJF algorithms, are known to minimize the expected total response time and expected makespan, respectively, when the jobs' execution times follow exponential distributions (Bruno et al. (1981)). A large body of work (see, e.g., Kleinberg et al. (1997); Goel and Indyk (1999); Möhring et al. (1999)) has been done to quantify the optimality and approximation under different objectives and arbitrary execution time distributions. Niño Mora (2009) gave a good survey of relevant results.

2.2 Current HPC scheduling in practice

The most commonly used resource managers in HPC are based on various implementations of the Parallel Batch System or more recently on Slurm (Yoo et al. (2003)). Reservation-based batch scheduling using priority queues and backfilling algorithms is the current de facto solution in implementing HPC schedulers. Most implementations use a four-step repetitive iteration algorithm in order to decide which jobs to run and on what resources. A new cycle is being triggered either by state changes, like starting, ending and modifying a job or by reaching a timeout. In the first step of the process, the scheduler communicates with the resource manager, such as Torque (Mukherjee et al. (2007)) or Slurm in order to update its information about the cluster's resources. Next, the scheduler decides which jobs to start, in what order and where they will be executed. Different policies are used to determine the exact method used, but in all cases jobs are ordered in one or multiple queues after which the scheduler attempts to execute them on available nodes or cores in the cluster. For example, the Moab (Capit et al. (2005)) scheduler calculates a priority for each submitted job that can later be adjusted by system administrators in order to target specific factors as being more important than others. The jobs are then placed in one queue ordered by the priority, after which Moab attempts to start the jobs, beginning at the top and moving down the queue. If a job cannot be started, Moab creates a reservation for the given job and starts the backfill algorithm to find small jobs in the queue that it will be able to start. If the chosen job can be started, Moab switches to the job placement algorithm by going through a simple elimination process: i) all nodes that do not have enough resources to run the job are eliminated from consideration (not enough memory or not enough cores); ii) all nodes that cannot run the job because of different policies are also removed (e.g. due to reservations); iii) the remaining nodes are sorted based on different priority requirements (e.g. given by administrators to equalize node usage) and the top nodes/cores are selected for execution. The last two steps in the scheduling process are used for refreshing reservations and updating statistics.

There is a continuous trade-off between overall system efficiency (increase cluster usage by scheduling primarily large jobs) and application response (the time jobs need to wait in queue to be executed). Different HPC schedulers use different policies to deal with this trade-off by giving higher priority to larger jobs and either using different backfilling algorithms to execute smaller jobs or by adjusting priorities depending on the time a job waited in the queue before being scheduled for execution. Users submit jobs specifying the amount of resources needed (number of nodes/cores as well as optionally the type of nodes and/or the amount of memory per core required by the application). The users must also provide the expected runtime for each submitted job. The scheduler takes all this information into account when setting the job priorities as well as when choosing the set of nodes for each execution. This workflow works well for scientific applications since the amount of resources needed is known in advanced with a fairly large probability and since there are enough small jobs submitted so that backfilling algorithms can hide the wasted cycles of the few less predictable jobs.

Clusters of commodity servers are currently a feasible alternative to major computing platforms. These servers/data centers use several computing frameworks, such as MapReduce (Dean and Ghemawat (2008)) or Dryad (Isard et al. (2007)), to simplify the usage of the cluster, each working with a variety of different resource managers for their job scheduling needs. The most common frameworks include Hadoop, which uses YARN (Vavilapalli et al. (2013)) as its default resource manager, and Apache Spark (Zaharia et al. (2016)), which comes with its own standalone resource manager or with YARN or Mesos (Hindman et al. (2011)). Unlike typical HPC resource managers, Mesos delegates control over scheduling to the underlying frameworks. It decides how many resources to propose to each framework depending on the requests and different policies (like fair sharing). The frameworks then decide which resources to accept and which tasks to run on them. Another example is YARN (Vavilapalli et al. (2013)), currently Hadoop's default resource manager. It decouples the programming model from the resource management infrastructure, and delegates it to application-level components. The global resource manager matches cluster state against the resource requirements reported by running applications. This allows YARN to enforce global scheduling properties like priorities for capacity or fairness purposes, but it generally requires the scheduler to obtain an accurate understanding of the applications' resource requirements. Decentralized scheduling models do not always lead to globally optimal scheduling solutions, but for workloads that consist of fine-grained tasks (as in MapReduce and Dryad), they typically show good performance.

High-level frameworks like E-HEFT (Samadi et al. (2018)) and Hive (Thusoo et al. (2010)) often treat a workflow of MapReduce jobs as a DAG, each filtering, aggregating, and projecting data at every stage of the computation. E-HEFT takes into account the variety and heterogeneity of virtual machines in a cloud computing cluster (e.g., different bandwidths, transfer rates, and processing capacities) and defines a schedule for running and placing tasks so it minimizes the total execution time and the unnecessary data transfers between virtual machines.

2.3 Scheduling challenges for unpredictable workflows

HPC schedulers rely on accurate estimates for the requested execution times. Reservation based scheduling, either by using SJF, LJF or batch priority based algorithms cannot deliver high performance when the estimates do not match the actual runtime of the applications. Backfilling algorithms are used to hide the wasted time caused by overestimation, but their performance depends on the amount of available small jobs in the waiting queue and on the size of the difference between estimates and actual execution times. Reservations guarantee fairness and global optimal performance only when the execution times are known in advance. For fields like neuroscience, this is not realistic (see Section 3 for a detailed analysis). In our recent work, we have started to develop reservation strategies for unpredictable (stochastic) workflows (Aupy et al. (2019)). This work is still in its early phase and currently only focuses on minimizing the expected makespan of a single job. Ultimately, we would like to combine these strategies with the ones developed in this new work. In this paper, we postpone the scheduling decision to the moment a resource becomes available in the system. Instead of using reservations to decide when an application will run, we combine greedy scheduling with the "on-the-fly" concept to propose novel scheduling solutions that can be applied to workflows whose execution times are unpredictable.

Task based schedulers as well as resource managers designed for clusters of commodity servers take advantage of the fact that their workloads consist of short tasks, and only need to reallocate resources when tasks finish. This reallocation happens frequently enough so that new frameworks and tasks acquire their share quickly and can start running without needing a reservation to ensure their completion. Neuroscience applications do not share properties with either HPC workloads or MapReduce tasks. They typically run on a small number of processing units, unlike HPC workloads that take advantage of the large number of cores offered by an HPC machine. At the same time, their execution takes large periods of time which makes them less suited for MapReduce frameworks. In addition, their resource requirements are wildly variable throughout their execution and among instances of the same application. In this paper, we highlight the limitation of these methods and show the need for a new framework that can adapt to stochastic resource requirements and a new paradigm in exploiting HPC systems.

3 Characterization of neuroscience workflows

In contrast to typical scientific applications, neuroscience workflows are usually executed sequentially one after another, and the execution of one workflow may require the outputs of some previously completed ones. Moreover, these dependencies are not fixed and can change over time depending on the needs of the neuroscience community. In addition to the dynamic dependencies created between workflows, each workflow typically contains several stages, whose executions depend on the characteristics of the input data, such as size, image quality or similarity to some internal parameters. The modular nature of neuroscience code both within and between workflows can make the execution time and input/output data traffic of the different instances vary by several orders of magnitude. In this section, we characterize the variability in the resource requirements of neuroscience workflows and highlight their key differences with typical scientific applications that create performance bottlenecks when running on current HPC systems. Based on this characterization, we then create several sets of synthetic workloads that will be used to evaluate the performance of different scheduling schemes for these emerging workflows.

3.1 Variability in resource requirements

Figure 1 shows the variability in the I/O traffic and walltime of two neuroscience workflows, namely, a whole brain segmentation workflow (also known as Multi-Atlas; Asman and Landman (2014)) and a diffusion QA and pre-processing workflow (also known as dtiQA_v2; Lauzon et al. (2013)). The figure shows 500 runs for each workflow within a period of 6 months (from September 2014 to February 2015) on the Vanderbilt high-performance computing cluster called ACCRE. In both workflows, there are large variations in the resource requirements, both in execution time (from 11 to 45 hours) and in the amount of data generated (from a few MBs to tens of GBs for example). We believe such variability is due to a combination of code-level changes (over larger periods of time) and variations in the input data (for instances closer in time).

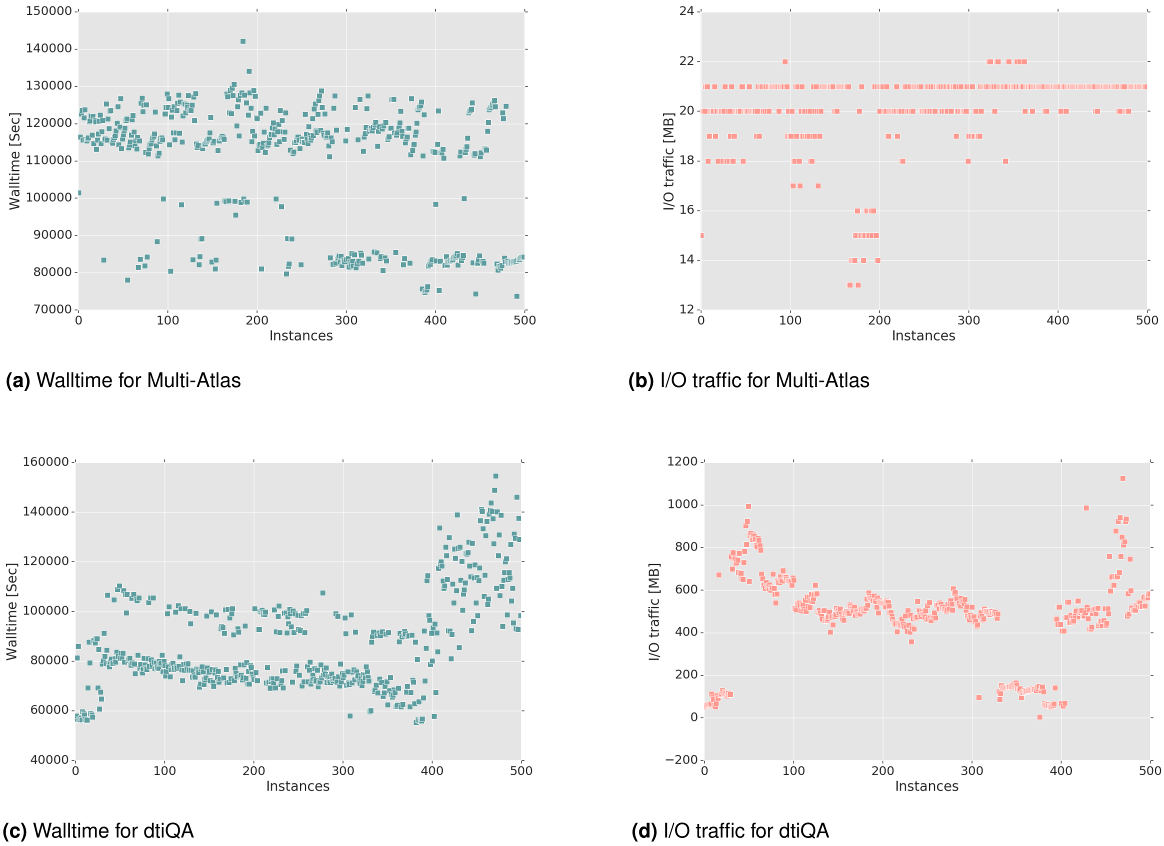
In fact, the highly variable resource requirements shown in Figure 1 represent a common phenomenon that is observed for many workflows used by the neuroscience community. Figure 2 plots the aggregate information about the resource requirements for 31 representative neuroscience workflows (Harrigan et al. (2016)) run on the ACCRE cluster between 2013 and 2016. We can see that most of these workflows show a large variation with more than one order of magnitude in both execution time and I/O traffic. Even workflows with smaller variations, such as the Generic fMRI workflow that implements a functional connectivity analysis method, still show around 20% variation for its execution time and over 10% variation for the I/O traffic. Furthermore, workflows that share similar execution time characteristic do not necessarily share similar I/O characteristic, and vice versa. For example, the Multi-Atlas and dtiQA workflows have similar walltime patterns, ranging from 8-10 hours to over 40 hours, both with an average of aprox 70 hours. However they show very different I/O traffic patters, the Multi Atlas workflow has a small variation of only around 10% (in the order of tens of MBs) while the dtiQA workflow can generate as traffic between a few MBs to GBs of data depending on the input data it receives.

3.2 Unpredictability of resource requirements

Being able to schedule jobs with accurate predictions of their expected walltimes is the cornerstone of high-performance computing. Backfilling algorithms can be used to fill the gaps when the applications' resource requirements are over-estimated. However, backfilling only works if there are enough small jobs, and even in that case, it can be inefficient when there is a large difference between the estimate and the actual execution time.

In this section, we focus on quantifying the unpredictable nature of neuroscience workflows' resource requirements. We show that the traditional complexity analysis breaks down for these workflows due to a lack of correlation between the input size and the execution time. We then provide some possible explanations on the source of such unpredictability.

Time complexity analysis is commonly used in computer science to measure or estimate the worse case running time of a program or an algorithm. In general, the execution time $T(n)$ of an algorithm can be expressed as a function of its input size n using the big-O notation, which provides a mathematical tool to formally describe the asymptotic behavior of an algorithm's running time with the growth of its input size. Figure 3 shows the execution times predicted using linear regression for two well-known algorithms,



(a) Walltime for Multi-Atlas

(b) I/O traffic for Multi-Atlas

(c) Walltime for dtiQA

(d) I/O traffic for dtiQA

Figure 1. The amount of data generated per core and the walltime for 500 instances of a whole brain segmentation workflow (Multi-Atlas) and a diffusion QA and pre-processing workflow (dtiQA_v2) running on the ACCRE cluster between September 2014 to February 2015. The instances are ordered by their submission time. The scatter plots show a great variability, from a few MB to several GB for the I/O traffic and from a few hours to more than 40 hours for the walltime.

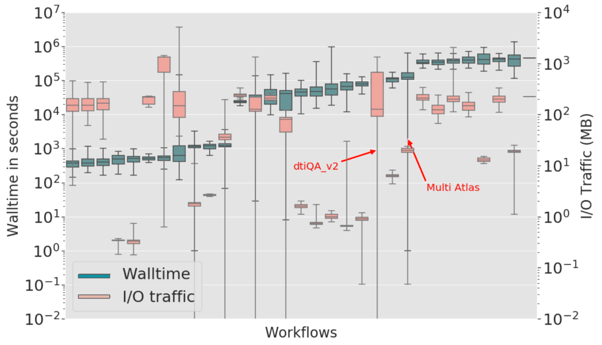


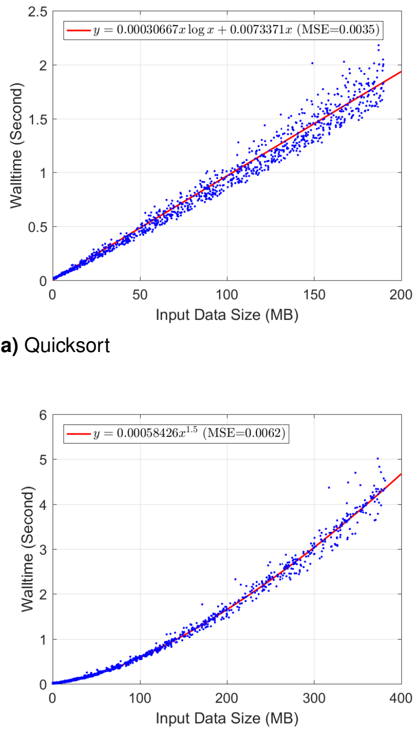
Figure 2. Range of walltime (in red) and I/O traffic (in green) for 31 representative neuroscience workflows running on the ACCRE cluster between 2013 and 2016. The workflows are sorted by their median walltime. Vertical lines show the intervals between the minimum walltimes and the maximum walltimes, bins show the 10th to 90th percentiles, and the horizontal lines highlight the medians.

quicksort and dense matrix-matrix multiplication, based on a number of sample runs with different input sizes. Note that for these two algorithms, a complexity analysis gives us the worse case complexity (respectively $O(n \log n)$ and $O(n^{1.5})$[*]). As expected, when the size of the input data is n, the time complexities for the two algorithms follow the prediction. In addition, one can notice the robustness of the complexity of these algorithms to their input data. Indeed in both case the mean squared errors (MSE) to the complexity functions are small (< 0.01).

In an attempt to similarly quantify the time complexity of neuroscience workflows, we looked at a number of commonly used functions up to the cubic order of input size: $n \mapsto 1$, $n \mapsto \log n$, $n \mapsto n$, $n \mapsto n \log n$, $n \mapsto n^2$, $n \mapsto n^2 \log n$, $n \mapsto n^3$, and tried to use them to fit our execution datasets with different input size n. The execution time of an algorithm with complexity $O(f(n))$ can be roughly defined by the equation $a \cdot f(n) + b$. To obtain the constants hidden behind the big-O with the best fitting curves, we ran a regression algorithm using all the six complexity classes above for each neuroscience workflow and chose the one that gives the smallest mean squared error. Note that in when discussing complexity, generally one consider the worse case complexity. However here we interpolate the average complexity since we want to study how predictable the execution time can be. We show that even in this “nicer” setup, the algorithms are not data-robust.

Out of all the workflows considered, only five workflows have relatively high correlation (> 0.7) between the observed execution time and the best execution time obtained

[*] For multiplication of two $k \times k$ matrices, the input size is $n = O(k^2)$ and the execution time is $T = O(k^3) = O(n^{1.5})$.



(a) Quicksort

(b) Matrix-matrix multiplication

Figure 3. Fitted curves for the running times of (a) quicksort (b) dense matrix-matrix multiplication with different input sizes. Blue dots represent an execution of an randomly generated instance with the corresponding input size and walltime. Red line represents the fitted curve using linear regression.

Table 1. Neuroscience workflows that present either a high correlation or a low MSE between the observed execution time and the interpolatation "execution time as a function of input data".

Workflow Name	**Workflow Description**	**Correlation**	**MSE**
fMRIQA_v2	Functional QA and pre-processing	0.82	3.95
MAGM_Normalize_v1	Fusion of structure and diffusion	0.06	5.81
Bedpost_v1	Probabilistic diffusion tractography	0.89	1861
pasmri_v1	Diffusion model fitting	0.88	13264
dtiQA_v2	Diffusion QA and pre-processing	0.64	1049.76
Multi-Atlas	Whole brain segmentation	0.04	3795.97

with the interpolation. Only three workflows have a MSE lower than ten. Table 1 presents details for the two best correlation workflows (Bedpost and pasmri), two best MSE workflows (fMRIQA and MAGM), and two interesting workflows: Multi Atlas and dtiQA. Note that except for one workflow (fMRIQA_v2), which has a relatively low error (MSE = 3.95) and high correlation (Pearson value of 0.82), the others do not show a good fit.

Figures 4 shows the scatter plots for three neuroscience workflows: the one with high correlation value and the lowest MSE (the fMRIQA_v2, a QA for functional magnetic resonance imaging code); and the workflows presented in Figure 1: the Multi-Atlas whole segmentation code and the diffusion QA and pre-processing workflow; and their respective best fitted curve. Each blue dot represents a running instance from the logs with the corresponding execution time and input data size, and the red line represents the curve given by the best fitting complexity class ($n \log n$, constant and $n^2 \log n$ respectively in this particular cases).

Generally speaking, the traditional complexity analysis breaks down for neuroscience workflows. One possible explanation is that the time complexity for these workflows may be dictated by both the "size" and some measure of "quality" of the input data. For example, the Multi-Atlas workflow uses a statistical label fusion algorithm to achieve brain segmentation by fusing 10 to 25 registered atlases iteratively under the expectation maximization (EM) fashion (Wang and Yushkevich (2013); Asman and Landman (2014)). It is conceivable that the execution time for a specific instance of the Multi-Atlas workflow could well depend on the input image quality and on how similar the input image is to the registered atlas entities, which go beyond the simple "size" of the input data.

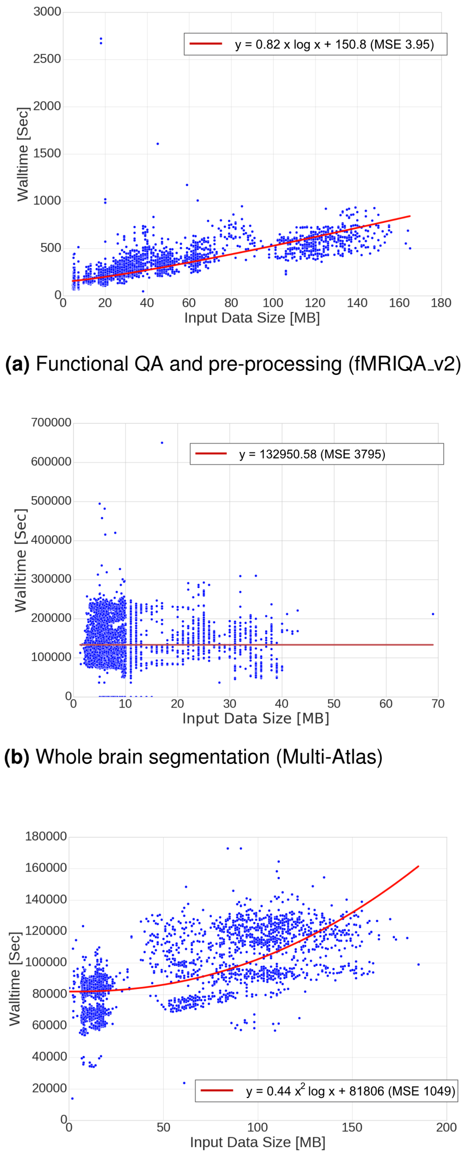
3.3 Performance vs. productivity

The HPC runtime system receives multiple neuroscience workflows submitted by several users at every given moment of time. Figure 5 presents the usage statistics per day for the neuroscience workflows during one year of ACCRE activity. An average day sees up to 5 to 10 different users submitting more than 100 workflows. Different days show different patterns with a large variety in the number of computational and I/O resources needed. Interestingly, a higher number of workflows executed in a day does not always correspond to a higher I/O traffic. While overall this pattern is not much different than running traditional scientific applications, HPC centers generally expect many small applications to accompany a few very large ones. In this regard, the traffic and computational variations are much more predictable since they are dictated by a few very large well-known HPC applications. In contrast, the neuroscience workloads contains a large variety of basic workflows, each with its own variable behavior.

The neuroscience community builds each workflow in several stages and chains several workflows together by requiring the input of one to be generated by others. Both stages as well as the dependencies between the workflows are dynamic and can change from one run to the next depending on each study's requirements. Unlike traditional scientific applications that are monolithic and tuned for giving the best performance on HPC systems, neuroscience workflows focus on productivity more than performance. Therefore, HPC systems need to be able to deal with highly dynamic workflows with unpredictable resource requirements.

4 Running unpredictable workloads on HPC systems

As we have seen in the previous section, traditional complexity analysis breaks down for neuroscience workflows and there seems to be no general rule that can be leveraged to model and predict their resource requirements. This poses a grand challenge for the HPC runtime systems, which rely on accurate estimations of an application's resource usage to make scheduling decisions. In this Section we try to demonstrate this challenge. One possible way to resolve this problem is to let the neuroscience community develop optimization and profiling tools that allow these workflows to adapt to the needs of current HPC systems. However, this solution will be expensive, since typical neuroscience



(a) Functional QA and pre-processing (fMRIQA_v2)

(b) Whole brain segmentation (Multi-Atlas)

(c) User-specific probabilistic tractography (dtiQA_v2)

Figure 4. Scatter plot of several runs for three functions and their best fitting functions ($n \log n$, $constant$, and $n^2 \log n$, respectively), the blue dots represent different runs of the given function with the corresponding execution times and input sizes, and the red line represents the best fitting curve.

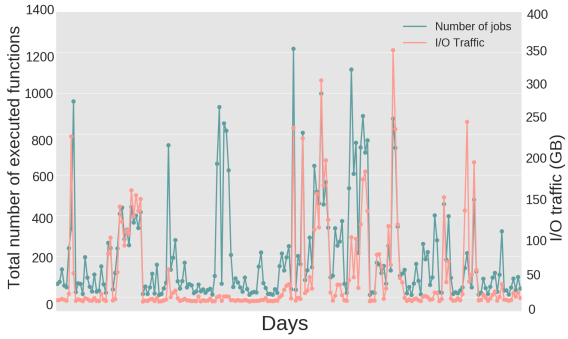


Figure 5. Number of functions and total I/O traffic per day for 7 months on the ACCRE system (system utilization is 84% on average).

workflows are in constant change of development with multiple modules implemented in different programming and scripting languages. Any workflow-specific optimization will inevitably involve ad-hoc strategies, and will be generally hard to debug and profile.

Instead of application-level optimizations, we consider adapting the scheduling solution in the runtime system to cope with the resource variability and unpredictability in neuroscience workflows. To that end, we use simulation to mimic the behavior of reservation-based batch scheduling, which is widely used in runtime systems such as the ones in ACCRE and other HPC centers. Additionally, we simulate two simplified “on-the-fly” schedulers that do not make reservations and instead schedule neuroscience workflows dynamically as resources become available. The goal of our simulation is to highlight the potential performance improvements that can be achieved for these unpredictable workloads instead of the performance bottleneck as is currently imposed by the scheduling solutions.

In this section, we first describe how we generate workloads that have behaviors close to neuroscience workloads (Section 4.1). Then, in Section 4.2, we present our implementation of the schedulers used for the study. We describe different metrics for the evaluation in Section 4.3 before comparing the different schedulers using our synthetic workloads in Section 4.4.

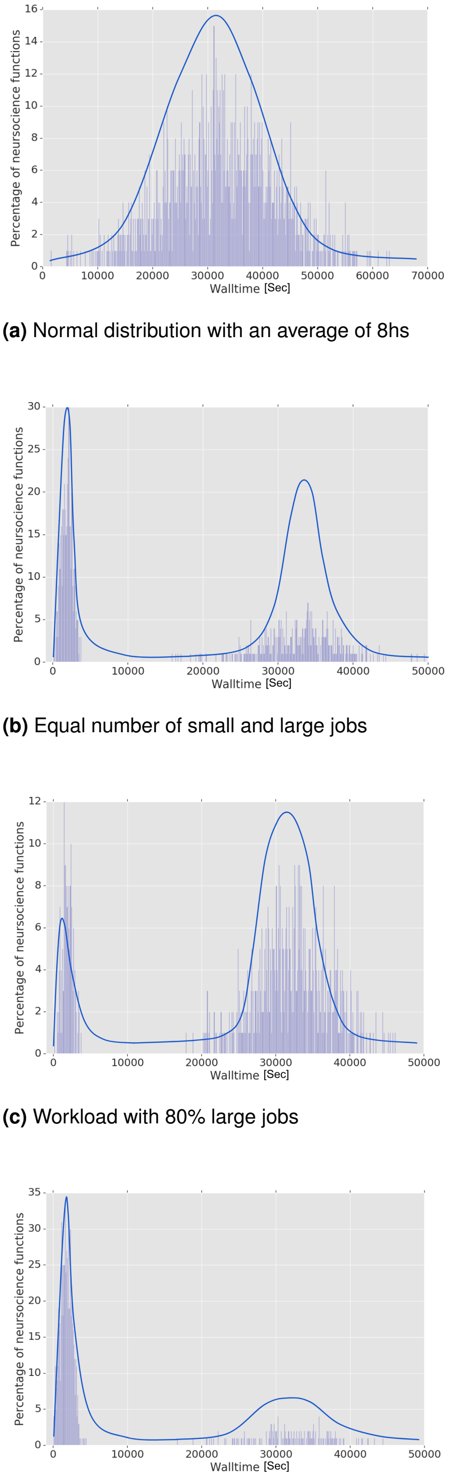
4.1 Synthetic workload generation

To better understand the performance of the current HPC schedulers, we created synthetic workloads based on our characterization of the neuroscience workflows presented in Section 3.

First of all, we generate jobs that all require the same number of computing nodes, which are equivalent to generating sequential jobs each executed on a single node. While this behavior is not typical for traditional HPC applications, it is common for neuroscience workflows, which tend to demand a fixed and small number of cores (e.g., less than 100) for efficiency purpose. Additionally, we extract the runtime behavior of these workflows by constructing four execution time distribution patterns as presented in Figure 6.

In the first pattern (Figure 6a), the execution time of the workloads is normally distributed with an average of 8 hours spanning from a few minutes up to 15 hours. The other three patterns divide the jobs into two categories, small and large, with the execution time of small jobs ranging from a few minutes to one hour and that of large job ranging from 3 to 15 hours. The second pattern (Figure 6b) has an equal amount of small and large jobs submitted over time in the system. The third pattern (Figure 6c) has 80% of the jobs in the system being large, while the forth pattern (Figure 6d) has 80% of the total jobs being small.

While analyzing the execution logs from the ACCRE cluster, we noticed that, on average, around 16% of the total neuroscience workflows were underestimated, requesting around 90% of their needed execution times (this number is found in the logs after resubmission). For all the other job submissions, workflows are typically overestimated, requesting in average 20% more than the actual execution



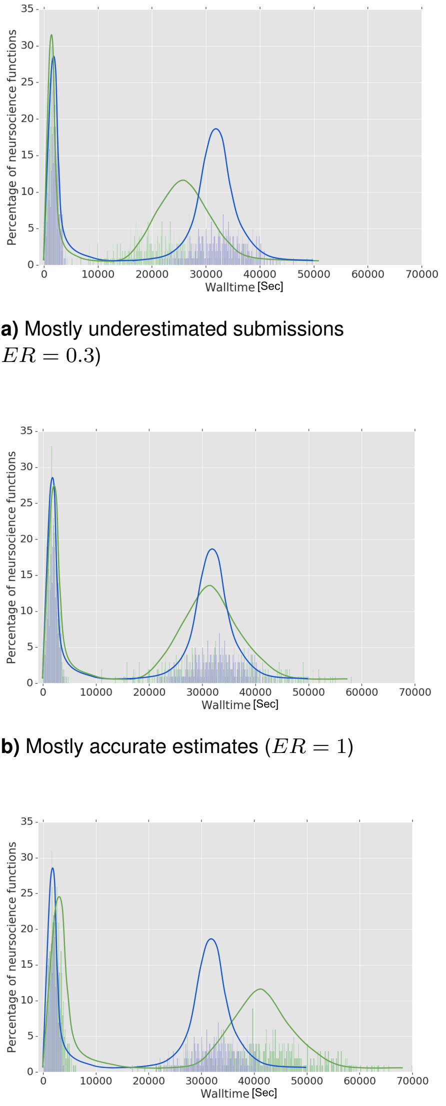
(a) Normal distribution with an average of 8hs

(b) Equal number of small and large jobs

(c) Workload with 80% large jobs

(d) Workload with 20% large jobs

Figure 6. Job size distributions for four types of simulated workflows.



(a) Mostly underestimated submissions $(ER = 0.3)$

(b) Mostly accurate estimates $(ER = 1)$

(c) Mostly overestimated submissions $(ER = 1.3)$

Figure 7. Job size distributions (blue) and estimated execution time distributions (green) for three values of the estimation ratio.

times. Let us define the *estimation ratio (ER)* to be

$$ER = \frac{\textit{estimated execution time}}{\textit{actual execution time}} \tag{1}$$

To simulate the above trend on execution time estimation, we created a random variable ER for estimation ratio that follows a normal distribution. The mean of ER is $\mu_{ER} = 1.2$ and we chose a standard deviation so that 10-15% of the generated values are below 1. In practice, this corresponds to a standard deviation $\sigma_{ER} = 0.2$.

To understand the effects of overestimation and underestimation on job scheduling, we try different values for the mean μ_{ER}: from 0.5 to 1.7. For example, $\mu_{ER} = 1$ means that most jobs will have an estimated time close to the actual execution time. In this case, the number of submissions with underestimated execution times will on average be equal to the number of submissions that overestimate the execution times (for normal distribution the median is equal to the mean). In case of underestimation in the RBS scheme, almost all jobs after the first resubmission will succeed since the increased estimation will likely exceed the actual execution time. On the other hand, small values for μ_{ER} (e.g., 0.5) will result in a high number of jobs being underestimated with a requested time equal to 30% to 70% of the actual execution time. In this case, most jobs might need to be resubmitted more than once when scheduled by RBS in order to complete successfully. In contrast, large values for μ_{ER} (e.g., 1.5) intuitively lead to more jobs having overestimation in execution time, thus completing on their first submissions. We show the estimation execution time distribution in comparison to the walltime distribution of our generated workloads in Figure 7. Illustrated is the second workload pattern from Figure 6.

We construct workloads by using the four execution time distribution patterns from Figure 6 with different estimated times by changing the ER value as defined in Equation (1).

Job submissions follow the Poisson process with an inter-arrival time that follows the exponential distribution with a mean of a 8 minutes, as observed on the ACCRE cluster.

4.2 Simulated schedulers

We implement three schedulers: one reservation-based batch scheduler (RBS) as currently used by many runtime systems in HPC centers, and two "on-the-fly" schedulers that use the shortest estimated job first (SEJF) and longest estimated job first (LEJF) policies, respectively, coupled with imperfect resource estimation. The following describes the principles of these three schedulers in more detail.

First, the reservation-based scheduler RBS follows the steps described in Section 2. It gives high priorities to long jobs or jobs that have waited in the queue for more than a threshold (e.g., every 20 minutes the priority is increased) and uses backfilling algorithms to schedule short jobs that have low priorities. Each job is required to have a requested time provided by the user upon submission. RBS uses the requested times to determine when computing nodes will become available and schedules reservations for the high-priority jobs (e.g., the first 100 jobs in the queue based on their priorities). Specifically, for each high-priority job, a time slot corresponding to its requested time is reserved on the least-loaded node according to the existing reservations and their respective requests. Small jobs also receive reservations towards the end of the schedule due to their low priority; they can also be scheduled opportunistically in the gaps that are created by jobs that finish sooner than their requested times. Any running job whose actual execution time is higher than the requested time will be killed by RBS and needs to be re-submitted by the user with a larger request. A new schedule is triggered whenever there are available resources in the system or when a new job is submitted.

On the other hand, the two "on-the-fly" schedulers do not create reservations. Instead, they schedule jobs dynamically in an online manner as resources become available. Specifically, all submitted jobs are first placed in a queue with priorities determined by their requested times. The SEJF policy assigns higher priority to short jobs and the LEJF policy assigns higher priority to long jobs. Each time a computing node becomes available, for example, due to a job finishing or when the node completes maintenance cycles, the job with the highest priority in the queue is selected for execution on the node. Note that both SEJF and LEJF only use the jobs' requested times for assigning priorities and for selecting the job to execute next, but not for making resource reservations in the system, thus no job will ever be killed. The two "on-the-fly" schedulers are inspired by the SJF and LJF policies in online scheduling with known job execution times (See Section 2.1), which are known for optimizing the application-level performance (e.g., job response time) and system-level performance (e.g., makespan or utilization), respectively. The difference here is that the estimated/requested times of the jobs are used instead of the actual execution times, which are unknown to the schedulers beforehand.

We point out that the reservation-based scheduler is designed with the principle to balance application-level and system-level metrics. Thus, for traditional HPC applications, its performance is expected to lie in between that offered by SJF and LJF in terms of metrics, such as average response time and utilization. For neuroscience workflows that have highly variable and unpredictable resource requirements, the reservation-based scheduler might show worse performance for both application-level and system-level metrics compared to the two "on-the-fly" scheduling schemes. Section 5 presents simulation results that compare the performance of the three schedulers on neuroscience workloads.

4.3 Evaluation metrics

We will now formally define the metrics for evaluating the performance of the schedulers. We first introduce some notations with regard to the jobs and their schedules. Consider an HPC system that has m identical computing nodes. A set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ of n jobs is submitted to the system over time. Each job $J_j \in \mathcal{J}$ is characterized by three parameters: the submission (or arrival) time s_j, the requested processing time e_j and its actual execution time p_j. For the reservation-based scheduler, if $p_j > e_j$, then the job fails and needs to be resubmitted. For simplicity of simulation, we assume that job resubmission is automated and does not require human intervention. Once a job fails, it is automatically placed in the waiting queue again with a new larger requested time. The total running time of the job in the system, denoted by t_j, includes all the time the job has run, including the failed and successful executions. A job completes after the first successful execution with a completion time of c_j. For each job J_j, we define the following metrics:

- *Response time* $r_j = c_j - s_j$: the total time elapsed between the job's submission and completion;
- *Stretch (or slowdown)* $d_j = r_j/p_j$: the ratio between the response time and the actual processing time.

In particular, the stretch metric proportionately relates a job's total elapsed time in the system (due to actual execution, waiting in the queue and failures) to its actual processing demand, which better reflects the user's psychological expectation (i.e., user is willing to wait longer for larger jobs). Therefore, stretch is often considered a fairer metric compared to the simple response time metric and has been favored by recent studies.

We simulate scheduling scenarios using different neuroscience workloads generated in Section 3 to better understand the performance of different scheduling schemes. For this purpose, we monitor the performance metrics to characterize the efficiency at both user level and system level. From the users' perspective, we consider the average stretch of all jobs, and from the system's perspective, we consider the utilization of resources. Both metrics are defined below:

$$\text{Average job stretch: } D = \frac{1}{n}\sum_{j} d_i \tag{2}$$

$$\text{System utilization: } U = \frac{\sum_j p_j}{m(\max_j c_j - \min_j s_j)} \tag{3}$$

In the simulation, we make 10 runs for each scheduling scenario by randomly generating jobs from the synthetic probability distributions. We then report the average performance metrics presented above for the three considered schedulers.

4.4 Performance evaluation

In this section, we evaluate the performance of the reservation-based scheduler (RBS) and the two "on-the-fly" schedulers (SEJF, LEJF) (see Section 4.2) on neuroscience-type of workloads with variable and unpredictable resource requirements that we generated in Section 4.1.

As we have seen, for such workloads, the requested time generally depends on domain knowledge. Without a robust way of estimating their execution times, users typically resort to ad-hoc methods that suite the particular application domain. For instance, the Medical-image Analysis and Statistical Interpretation (MASI) Lab[†] at Vanderbilt uses the the average walltime of the last few runs of a neuroscience workflow as its estimated time. If the submitted job has a timeout failure, the estimated time is increased by 30-50% from the previous request. Based on the execution log we analyzed, this method causes more than 10% of all job submissions to fail due to insufficient time request. From the user's perspective, choosing smaller estimated execution times leads to higher probabilities of job failures while choosing larger estimations likely incurs longer waiting times and/or higher cost of using the resources.

Figure 8 presents the utilizations of the system for the four workloads while varying μ_{ER} from 0.5 to 1.7. All the workloads contain a total of 800 functions and a simulation time window between 24 and 48 hours depending on the inter arrival time. First, we observe that the two "on-the-fly" schedulers have similar performance with a small variation in utilization as we change the values of μ_{ER}. This is because the estimated times are only used for ordering the jobs and not for reservations, and they are generally proportional to the actual execution times. The best system utilization (as achieved by the LEJF scheduler) varies between 45% and 90% for the four workloads, depending on the execution time distribution and the ratio between large and small jobs.

The RBS scheduler has a lower system utilization between 20% and 65%, due to a large number of incurred failures (up to 1600) which are considered wasted time. As the estimated execution time increases (higher μ_{ER} values), the number of failures reduces and more jobs finish successfully. However, larger estimations create more gaps in reservations caused by the difference between the overestimated time and the actual time. These gaps represent wasted time as well, unless they are filled by small jobs using the backfilling algorithm. For each workload, there is a trade-off between the time wasted due to failures and the time wasted due to the gaps caused by overestimation and not having enough small jobs to backfill. The highest utilization is reached when the total number of failures falls between 50 and 200 (which represents 6% to 25% of total submitted jobs) for the workload containing mostly large jobs (Figures 8a and 8c). After this point, not enough small jobs are available to fill the gaps caused by the overestimations. Workloads with many small jobs (Figures 8b and 8d) do not show a decrease in utilization even under high execution time overestimation (e.g., with $\mu_{ER} \geq 1.5$). Overall, in all cases, the typical HPC scheduler (represented by the batch scheduler with an ER of 1.2) shows a decrease in utilization of 35% for workloads with a large number of small jobs and a decrease of 45% when large jobs dominate the submission queue.

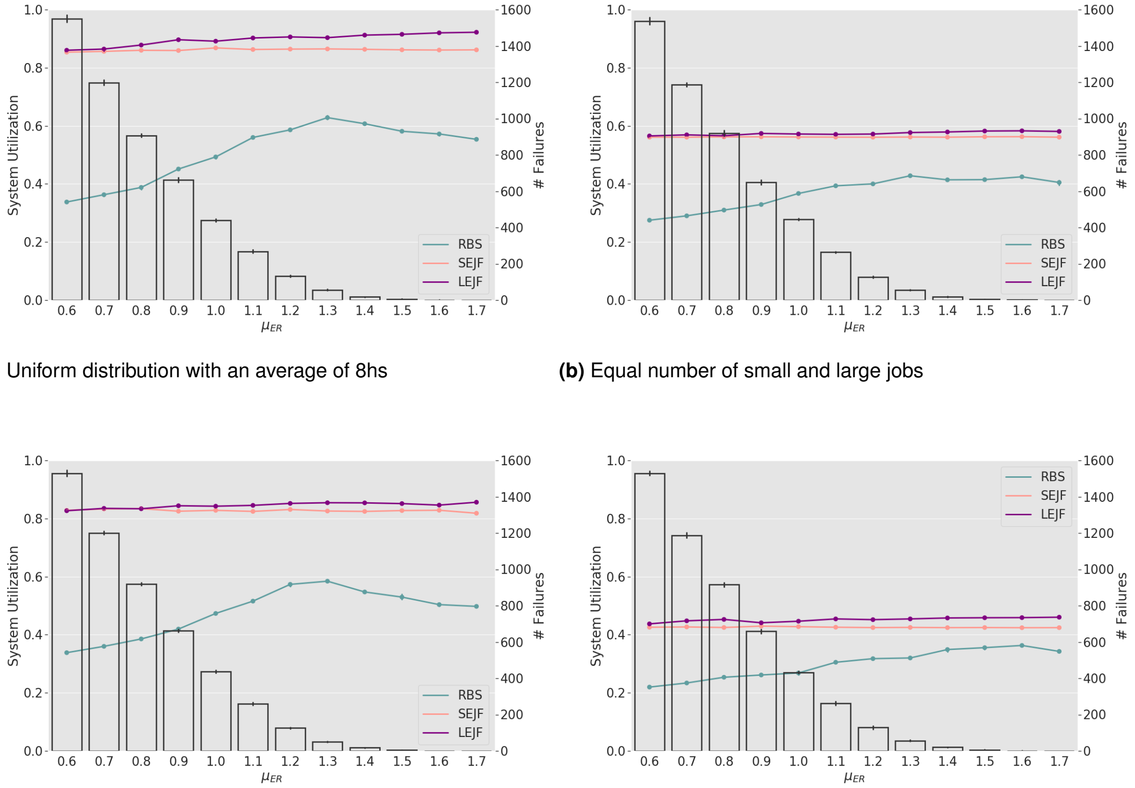
Figure 9 presents the average stretch of the jobs for the four workloads while varying μ_{ER} from 0.5 to 1.7. Recall that the stretch of a job represents the ratio between the job's response time (including all failed runs) and its actual execution time. We can see that the two "on-the-fly" schedulers have an average stretch between 1 and 3, suggesting that jobs complete within 3 times of their actual execution time. Additionally, the two schedulers do not show high variation in the average stretch for different μ_{ER} values since they do not incur any failures. In particular, the SEJF scheduler, by favoring small jobs, tends to reduce the average waiting time of the jobs, which translates to lower average stretch, since the longer waiting times for the large jobs are hidden by their high execution times. For LEJF, large jobs are favored and small jobs are delayed from starting their executions. Hence, the stretches for the small jobs will increase substantially compared to the decrease in stretch for the large jobs. This is visible in workloads dominated by large jobs (Figures 9a and 9c), where there is a gap between the performance of SEJF and LEJF.

The RBS scheduler can leverage backfilling to partially hide the increase in waiting times for small jobs. However, its performance is generally worse compared to that of the "on-the-fly" schedulers. As the number of failures decreases, the average completion time for all jobs decreases, resulting in lower average stretch. If enough small jobs are available to the backfilling algorithm, then larger values of μ_{ER} will result in lower waiting times for these smaller jobs. In this case, Figure 9d shows that the performance of RBS can come close to that achieved by the "on-the-fly" schedulers as long as jobs request more time than needed and the system has enough small jobs to fill in the gaps caused by such overestimation. This trend will continue until the the backfilling algorithm runs out of small jobs, at which point the waiting time for the large jobs will begin to dominate the average stretch. This behavior can be observed in three out of the four workloads (Figures 9a, 9b and 9c), and we expect it to hold for the last workload as well if we keep increasing the estimation (μ_{ER}). On average, the RBS scheduler increases the average stretch by around 2x and the average waiting time by around 3x compared to "on-the-fly" schedulers.

5 Towards robust HPC schedulers

Current HPC schedulers require a good understanding of the resource needs of all applications in order to efficiently use the computational power of large-scale machines. They

[†] https://my.vanderbilt.edu/masi/



(a) Uniform distribution with an average of 8hs

(b) Equal number of small and large jobs

(c) Workload with 80% large jobs

(d) Workload with 20% large jobs

Figure 8. System utilization of the three schedulers under different average estimation ratios (μ_{ER}) for each of the four workloads. The lines use the left axes and represent the utilization (as the ratio of total time spent in useful computing over total simulation time on all nodes). The bars use the right axes and represent the number of job failures caused by resource underestimation under RBS.

are designed primarily for applications that run on many computing nodes for long periods of time while sharing the system with a few other small applications. In addition, HPC schedulers assume applications are submitted with accurate estimated processing times compared to the actual walltime and that failure impact is minimized by checkpointing.

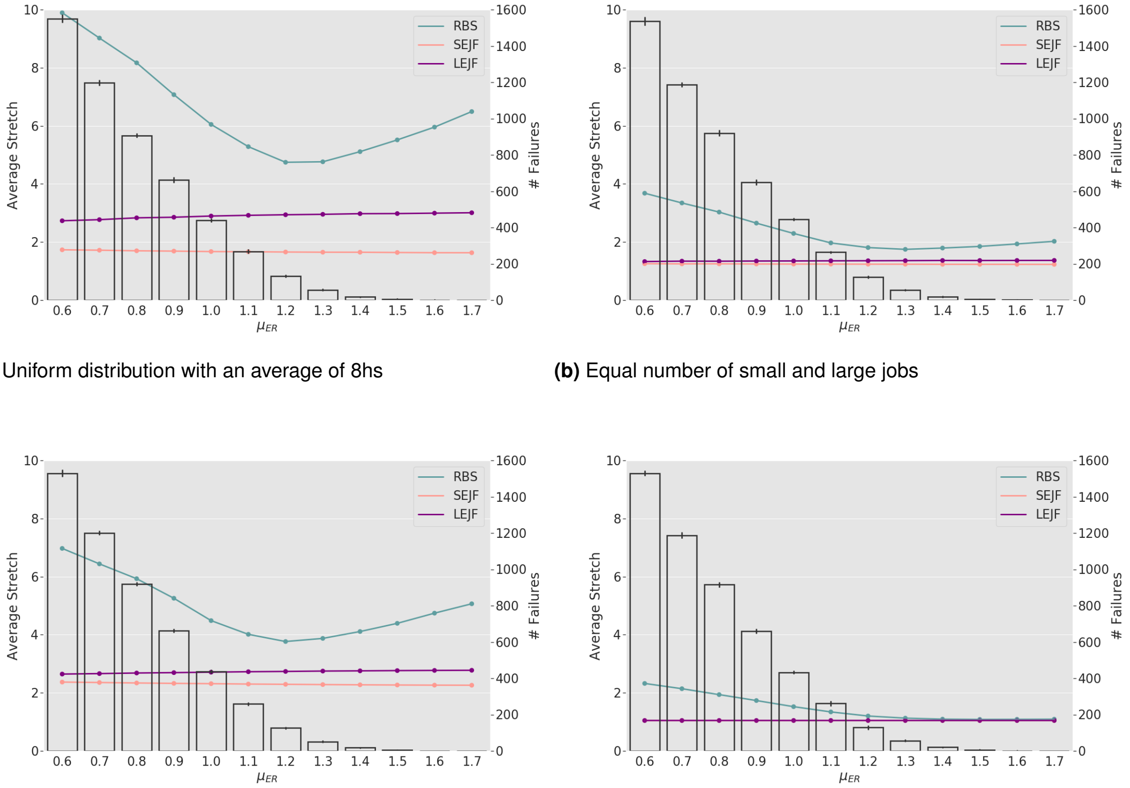
Neuroscience workflows show resource variability in all aspects of their execution, in the amount of computation time, of memory usage and of generated I/O. Their patterns change throughout their lifetime and are heavily application-dependent. In addition, their processing requirements are relatively uniform, with most of the parallelism being in the SPMD form. While each neuroscience workflow runs on a small number of computing nodes, the same workflow can be executed with very different input data (like the case of Multi Atlas that runs with different fMRI images).

Our experiments presented in Section 4.4 show that the current HPC scheduler reaches only up to 65% in system utilization compared to “on-the-fly” schedulers and have an average decrease of over 2 times in workflow stretch. In addition, if these workflows would run together with other scientific applications that generally use a large number of nodes and have long execution times, the neuroscience workflows will have even lower priority in the execution.

We next present two directions for adapting the current batch schedulers to the needs of unpredictable workloads such as the ones developed by the neuroscience community. One direction looks at workflow-level analysis in order to understand and contain their unpredictability and adapts the current schedulers without dramatically changing the paradigm. The second direction focuses on “on-the-fly” schedulers and how they can be implemented to deliver high performance to all types of workloads encountered by an HPC system.

5.1 *Workflow-level analysis*

In order to understand the degree of variability and unpredictability in performance within neuroscience workflows, we manually inspect a widely used neuroscience workflow, namely, Multi-Atlas (Wang and Yushkevich (2013); Asman and Landman (2014)), a whole brain segmentation code. Multi-Atlas is a well understood neuroscience workflow that is used as a preprocessing step for many other workflows (for example all functional connectivity analysis programs require the annotated brain generated by Multi-Atlas as an input). It is designed to automatically label objects of interest in fMRI images based on expert-labeled images. The technique uses multiple expert-segmented example images, called atlases, to register them to a target image. The deformed atlas segmentations obtained after the first step are afterwards combined using label fusion. The whole brain segmentation workflow used by the MASI lab uses a total of 15 expert-labeled images for each Multi-Atlas instance. These 15 images are chosen from a dataset of 45 existing expert-labeled fMRI images based on similarity to the target



(a) Uniform distribution with an average of 8hs

(b) Equal number of small and large jobs

(c) Workload with 80% large jobs

(d) Workload with 20% large jobs

Figure 9. Average job stretch of the three schedulers under different average estimation ratios (μ_{ER}) for each of the four workloads. The lines use the left axes and represent the average stretch (as the ratio of job response time over its actual execution time). The bars use the right axes and represent the number of jobs failures caused by resource underestimation under RBS.

image. Errors produced by label transfer for each of the 15 chosen images are reduced by label fusion that combines the results produced by all atlases into a consensus solution.

For each neuroscience workflow, two sources of variability need to be investigated: *ill conditioning* and *stability*. Ill conditioning characterizes the degree of variability caused by the properties of the input data, while stability measures the variability in execution time and resource consumption intrinsic to the algorithm being used, independent of the input data and parameters. In order to minimize the noise caused by the hardware in an HPC system, we ran Multi-Atlas on only one node using the same underlining hardware and found that the stability of Multi-Atlas is very high compared to the overall variability caused by different input data, with the code dependent variability of less than 10%. For the rest of the section, we will focus on analyzing the ill conditioning propriety of the workflow.

We analyzed logs that contain information about different Multi-Atlas runs on one node of the ACCRE cluster. After filtering out the killed and failed jobs as well as execution outliers, our database has 1011 instances of the Multi-Atlas runs, with the start and end times for the four phases of the workflow together with the list of the 15 atlases chosen for each instance. Among the four phases, the registration and fusion phases have the longest execution times in the order of hours while the pre- and post-processing phases run only for tens of minutes. There is some execution time variability in the pre- and post-processing phases. However, it is insignificant compared to the variation in the fusion phase. The registration has much less variation compared to the fusion phase and shows little correlation with the total execution time of the workflow. Figure 10 shows the walltimes of the registration and fusion phases in relation to the total execution times for each of the 1011 Multi-Atlas instances. The fusion phase can run for less than an hour to tens of hours and shows a strong correlation with the total execution time of Multi-Atlas.

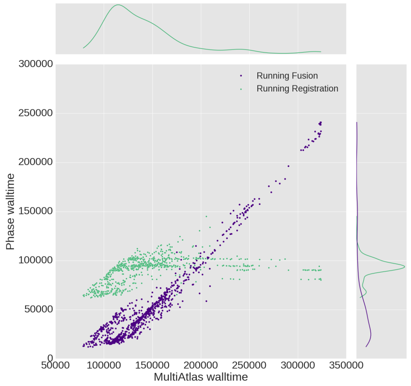


Figure 10. Correlation between the fusion and registration phases of the Multi-Atlas workflow in relation to the total execution time for each instance.

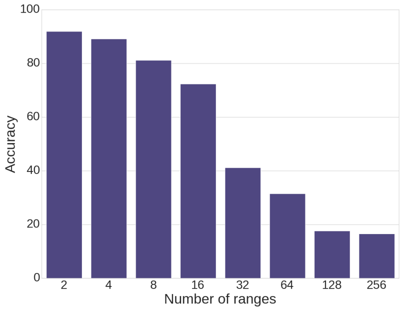


Figure 11. Accuracy (percentage of corrected predicted workflows over total number of workflows) for predicting ranges of performance.

The fusion phase of Multi-Atlas assigns labels based on the results given in the registration phase for each atlas in an iterative process. As long as the labels have changed compared to the previous loop, the algorithm uses a weighted vote on the label of each segment based on each atlas registration and adjust weight of each atlas based on current labeling. We analyzed the choice of the 15 atlases in connection to the walltime of each instance. Results show that the choice is not correlated to performance (there are instances that chose the same 15 atlases and show drastically different walltimes). Each atlas is chosen based on a similarity value to the target image given by the Pearson correlation. This similarity value indicates how close the target image is to the set of 15 altases used for registration. When adding this information into the analysis, we were able to predict ranges of performance with reasonable accuracy. In particular, we divide the overall execution time range of all Multi-Atlas instances into a number of smaller ranges.

The higher the Pearson correlation values between the target image and the atlases (regardless of which 15 out of the 45 are chosen), the better the performance of the fusion phase. Using this relationship, we could predict good or bad performance runs with good accuracy. Figure 11 shows the prediction accuracy depending on the number of ranges used (i.e., how narrow or wide the performance ranges are).

Despite the promise in the workflow-level analysis above, it requires tremendous effort to understand the performance bottlenecks. For Multi-Atlas, we used logs from over 1000 runs to extract behavior characteristics and several extra runs to test our theories. Yet it is only one of commonly used workflows with code that is relatively stable, straightforward and can be executed in a few hours. Other neuroscience workflows that require days to run or that are in constant development changes, as well as complex workflows that use multiple modules (some implemented in different programming languages) will be harder to analyze and understand. Overall, workflow-level analysis is possible, but continuous effort needs to be dedicated in order to keep up with the fast pace changes in code development.

5.2 Hybrid schedulers for HPC

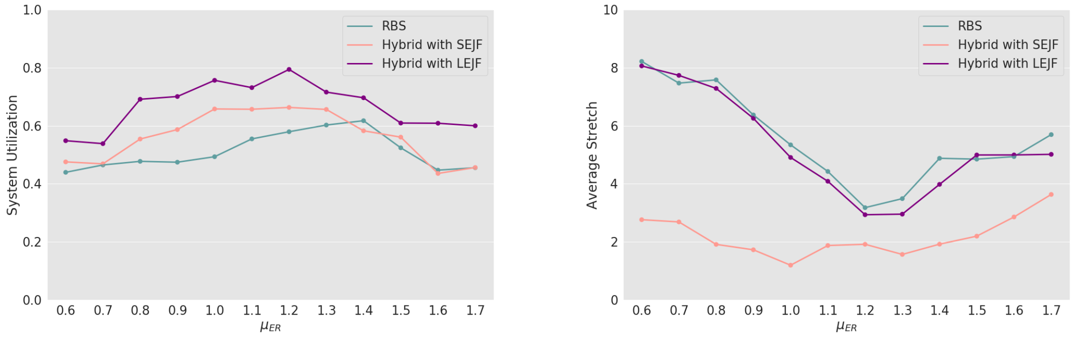
The results in the previous section show that “on-the-fly” schedulers can achieve better performance when dealing with neuroscience workflows due to the fact that these schedulers do not require accurate estimates for the job execution times. Neuroscience workflows generally use the capabilities of an HPC system in the SPMD mode, where each program runs on a small number of computing nodes. The uniformity in the use of computing nodes makes resource reservations unnecessary, thus allowing “on-the-fly” schedulers to work better on workloads with unpredictable resource requirements.

Reservation-based batch schedulers rely on backfilling to hide the wasted time caused by overestimation and cause job failures due to underestimation. Accurate estimates are the cornerstone for high performance. In addition, backfilling algorithms also require accuracy since they choose jobs based on the provided estimates. The literature has shown that the performance of backfilling algorithms is directly related to how well the execution times of small jobs can be estimated (Feitelson et al. (2005)).

Workflow-level analysis can be applied to contain the unpredictability in resource requirements for neuroscience workflows. Similarly to our analysis of Multi-Atlas, performance ranges can be used instead of accurate estimates to design new scheduling methods that can be integrated into current batch schedulers. However, hard-wiring workflows to optimize an application for a specific dataset and cluster architecture is hindered by the shift in architecture design for current HPC systems and will not serve a dynamic community like neuroscience.

“On-the-fly” schedulers do not require accurate estimates for the job processing times and can adapt to different workloads as long as all jobs require a similar number of computing nodes. While this requirement might seem strict, many of the current HPC systems already follow this trend. For example, out of the workloads that ran on the XSEDE systems (Towns et al. (2014)) in 2017, over 20% represented serial codes and more than half of the jobs ran on only one node with a significant proportion of them running on 32 or fewer processes/threads per node (Simakov et al. (2018)).

Commodity clusters are an alternative to traditional HPC systems and can use “on-the-fly” approaches to schedule tasks. These resource managers expect applications that can easily be decomposed into fine-grained tasks that execute for short periods of time. In addition, while MapReduce frameworks do not require execution time estimates from users, they do require resource usage (memory, storage, communication) in order to offer good performance. The resource requirements of neuroscience workflows change throughout their execution and cannot be predicted in advance. We believe MapReduce frameworks could be adapted to include the stochastic nature of emerging scientific workflows. The scheduling literature contains studies that have analyzed the way mixed workloads can be ran on HPC platforms (one such example is done by Ayyalasomayajula and Maschhoff (2016)). All these studies assume a mix of HPC and MapReduce type of workloads. The study by Ayyalasomayajula and Maschhoff (2016) used Mesos under which YARN is run for MapReduce workloads and a modified version of Slurm for Cray Graph Engine (CGE) that does real-time analytics for large and complex graph problems. Both frameworks are configured to interact with Mesos for acquiring resources to launch their jobs, which allows dynamic resource partitioning between the two. Slurm was



(a) System utilization

(b) Average workflow stretch

Figure 12. System utilization and average workflow stretch for a hybrid scheduler that combines "on-the-fly" scheduling and reservation-based batch scheduling.

extended to permit leasing of resources on an as-needed basis from Mesos controlled cloud computing cluster. The CGE analytics workloads consist of multiple small tasks that can be easily monitored and adjusted for performance either by killing a task that is running longer than expected or allowing space after the end of one task for a possible execution of another that shares data. We expect large codes that execute for hours and have a uniform resource consumption during this time will not perform well on such systems.

Either adding a framework for neuroscience-like workloads into Mesos or isolating them and designing "on-the-fly" schedulers that can be integrated into the current HPC runtime system could lead to improvements in both system-level and user-level performance metrics. For demostrative purposes, we envision in this section a hybrid solution that combines "on-the-fly" scheduling with reservation-based batch scheduling. Specifically, the hybrid scheduler monitors the resource requirements of jobs waiting in the queue and switches between two modes depending on their characteristics. If all jobs share similar processing characteristics, then "on-the-fly" schedulers can be used. Otherwise, soft reservations will be created while backfilling algorithms will opportunistically schedule the heterogeneous workflows whenever possible. These workflows need not guarantee the accurate estimates of their execution times and will be checkpointed if they run out of reservations. Indeed, checkpointing solutions are generally used to alleviate the resource wastage caused by underestimation. However, when using simple batch schedulers, each time a job is killed it needs to go through the waiting queue in order to be scheduled again. In our hybrid solution, the runtime is responsible for checkpointing and restarting a job as soon as possible. We simulated this hybrid approach by running a synthetic workload of 300 jobs with the distribution from Figure 6a each running on one node, with 30 large monolithic applications running on 50% of the machine. We used either SEJF or LEJF to implement the "on-the-fly" mode of the runtime system. The results, which are presented in Figure 12, show that our hybrid approach can improve both system utilization and average stretch in most scenarios.

It is important to note that while this paper focuses on neuroscience workflows, significant classes of non-monolithic applications share the same characteristics. The study by Weidner et al. (2016) looks at a range of applications that are driven by adaptive algorithms that can require different resources depending on their input data and parameters. Specifically, they show how the execution time of a Kalman-Filter application can differ by several hours depending on the model's initial conditions and how an AMR simulation of a molecular cloud might require additional compute time to increase the resolution in an area of interest.

Lastly, we point out that more sophisticated middleware solutions might yield even better results if HPC schedulers could be designed for workloads with variable and unpredictable resource requirements. "On-the-fly" scheduling offers an attractive paradigm that deserves further analysis, possibly in combination with other flexible solutions. For example, being able to schedule based on ranges of execution times instead of a fixed estimate will allow workflows like Multi-Atlas to achieve higher performance on HPC systems.

6 Conclusion

In this paper, we have analyzed an emerging type of scientific workflows with highly variable and unpredictable resource requirements using the field of neuroscience as an example. By analyzing their execution time characteristics, we have shown that reservation-based batch scheduling, which is currently implemented in HPC systems, cannot provide high performance, and that "on-the-fly" scheduling offers promising results in both system-level and user-level performance measures. We have illustrated how such "on-the-fly" scheduling can be integrated into the runtime solution in combination with reservation-based scheduling to form hybrid schedulers to cope with the workload heterogeneity for future HPC systems. In the future, we plan to use our preliminary results to investigate more complex scheduling methods that are adapted to the unpredictable nature of these applications without posing a negative impact on current large-scale scientific applications.

Acknowledgement

We thank the VUIIS Center for Computational Imaging for sharing de-identified logs without patient or investigator identifiable data. This research was supported in part by National Science Foundation grant CCF1719674 and Vanderbilt Institutional Fund.

References

Asman AJ and Landman BA (2014) Hierarchical performance estimation in the statistical label fusion framework. *Medical Image Analysis* 18(7): 1070 – 1081.

Aupy G, Gainaru A, Honoré V, Raghavan P, Robert Y and Sun H (2019) Reservation strategies for stochastic jobs. In: *IEEE International Parallel and Distributed Processing Symposium*.

Ayyalasomayajula H and Maschhoff K (2016) Experiences running mixed workloads on cray analytics platforms. *Cray User Group Conference* .

Becchetti L and Leonardi S (2004) Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. *Journal of the ACM* 51(4): 517–539.

Brucker P (2001) *Scheduling Algorithms*. 3rd edition. Berlin, Heidelberg: Springer-Verlag. ISBN 3540415106.

Bruno J, Downey P and Frederickson GN (1981) Sequencing tasks with exponential service times to minimize the expected flow time or makespan. *Journal of the ACM* 28(1): 100–113.

Capit N, Costa GD, Georgiou Y, Huard G, Martin C, Mounie G, Neyron P and Richard O (2005) A batch scheduler with high level components. In: *CCGrid*. pp. 776–783.

Chekuri C, Motwani R, Natarajan B and Stien C (1997) Approximation techniques for average completion time scheduling. In: *SODA*. New Orleans, LA, USA, pp. 609–618.

Conway R, Maxwell W and Miller L (1967) *Theory of Scheduling*. Addison-Wesley.

Dean J and Ghemawat S (2008) MapReduce: Simplified data processing on large clusters. *Commun. ACM* 51(1): 107–113.

Feitelson DG, Rudolph L and Schwiegelshohn U (2005) Parallel job scheduling — a status report. In: *JSSPP*. pp. 1–16.

Garey MR and Johnson DS (1990) *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co. ISBN 0716710455.

Goel A and Indyk P (1999) Stochastic load balancing and related problems. In: *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99. pp. 579–586.

Graham RL (1966) Bounds for certain multiprocessing anomalies. *Bell System Technical Journal* 45(9): 1563–1581.

Harrigan RL, Yvernault BC, Boyd BD, Damon SM, Gibney KD, Conrad BN, Phillips NS, Rogers BP, Gao Y and Landman BA (2016) Vanderbilt university institute of imaging science center for computational imaging XNAT: A multimodal data archive and processing environment. *NeuroImage* 124: 1097 – 1101.

Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz R, Shenker S and Stoica I (2011) Mesos: A platform for fine-grained resource sharing in the data center. In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11. pp. 295–308.

Isard M, Budiu M, Yu Y, Birrell A and Fetterly D (2007) Dryad: Distributed data-parallel programs from sequential building blocks. In: *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*. pp. 59–72.

Kleinberg J, Rabani Y and Tardos E (1997) Allocating bandwidth for bursty connections. In: *STOC*. pp. 664–673.

Lauzon CB, Asman AJ, Esparza ML, Burns SS, Fan Q, Gao Y, Anderson AW, Davis N, Cutting LE and Landman BA (2013) Simultaneous analysis and quality assurance for diffusion tensor imaging. *PLOS ONE* 8(4): 1–15.

Möhring RH, Schulz AS and Uetz M (1999) Approximation in stochastic scheduling: The power of LP-based priority policies. *Journal of the ACM* 46(6): 924–942.

Motwani R, Phillips S and Torng E (1993) Non-clairvoyant scheduling. In: *SODA*. Austin, TX, USA, pp. 422–431.

Mukherjee T, Tang Q, Ziesman C, Gupta SKS and Cayton P (2007) Software architecture for dynamic thermal management in datacenters. In: *2007 2nd International Conference on Communication Systems Software and Middleware*. pp. 1–11.

Niño Mora J (2009) Stochastic scheduling. *Encyclopedia of Optimization* : 3818–3824.

Pinedo ML (2008) *Scheduling: Theory, Algorithms, and Systems*. Third edition. Springer-Verlag New York, Inc.

Pruhs K, Torong E and Sgall J (2004) Online scheduling. In handbook of scheduling: Algorithms, models, and performance analysis, Chapter 15, CRC Press .

Samadi Y, Zbakh M and Tadonki C (2018) E-HEFT: Enhancement heterogeneous earliest finish time algorithm for task scheduling based on load balancing in cloud computing. *2018 International Conference on High Performance Computing and Simulation (HPCS)* : 601–609.

Shmoys DB, Wein J and Williamson DP (1991) Scheduling parallel machines online. In: *FOCS*. San Juan, Puerto Rico, pp. 131–140.

Simakov NA, White JP, DeLeon RL, Gallo SM, Jones MD, Palmer JT, Plessinger BD and Furlani TR (2018) A workload analysis of NSF's innovative HPC resources using XDMoD. *CoRR* abs/1801.04306.

Thusoo A, Sen Sarma J, Jain N, Shao Z, Chakka P, Zhang N, Anthony S, Liu H and Murthy R (2010) Hive - a petabyte scale data warehouse using hadoop. pp. 996–1005.

Towns J, Cockerill T, Dahan M, Foster I, Gaither K, Grimshaw A, Hazlewood V, Lathrop S, Lifka D, Peterson GD, Roskies R, Scott JR and Wilkins-Diehr N (2014) XSEDE: Accelerating scientific discovery. *Computing in Science and Engineering* 16(5): 62–74.

Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, Saha B, Curino C, O'Malley O, Radia S, Reed B and Baldeschwieler E (2013) Apache hadoop yarn: Yet another resource negotiator. In: *Proceedings of the 4th Annual Symposium on Cloud Computing*. pp. 5:1–5:16.

Wang H and Yushkevich P (2013) Multi-atlas segmentation with joint label fusion and corrective learningan open source implementation. *Frontiers in Neuroinformatics* 7: 27.

Weidner O, Atkinson M, Barker A and Filgueira Vicente R (2016) Rethinking high performance computing platforms: Challenges, opportunities and recommendations. In: *Proceedings of the ACM International Workshop on Data-Intensive Distributed Computing*. pp. 19–26.

Yoo AB, Jette MA and Grondona M (2003) Slurm: Simple linux utility for resource management. In: *JSSPP*. pp. 44–60.

Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ, Ghodsi A, Gonzalez J, Shenker S and Stoica I (2016) Apache spark: A unified engine for big data processing. *Commun. ACM* 59(11): 56–65.