# Adaptive Memory Allocation Model in Multi-Core Machine Clusters

Roman Malih
Ben-Gurion University
1 Ben-Gurion Ave.
Beer-Sheva 8443944, Israel

Ofer Levi
The Open University of Israel
1 University Road
Raanana 4353701, Israel

Diamanta Benson-Karhi
The Open University of Israel
1 University Road
Raanana 4353701, Israel

## ABSTRACT

In this work, we address the problem of robust real-time scheduling and resource allocation in real-life, complex environments with unpredictable stochastic behavior. We focus on an important, simplified case study from the computing domain, i.e., memory resource allocation and job scheduling on a dedicated computer cluster with shared memory. We explore techniques of resource utilization given a defined computing environment, and we develop an adaptive model to handle incoming computing jobs. We test and validate our proposed model by simulation using the Matlab and Sim Events software packages. An adaptive model, designed for a cluster of dual-core machines with shared memory constraints, is proposed. We have shown that our model is efficient and robust despite making no assumptions about the stochastic characteristics of the incoming jobsIn this work, we address the problem of robust real-time scheduling and resource allocation in real-life, complex environments with unpredictable stochastic behavior. We focus on an important, simplified case study from the computing domain, i.e., memory resource allocation and job scheduling on a dedicated computer cluster with shared memory. We explore techniques of resource utilization given a defined computing environment, and we develop an adaptive model to handle incoming computing jobs. We test and validate our proposed model by simulation using the Matlab and Sim Events software packages. An adaptive model, designed for a cluster of dual-core machines with shared memory constraints, is proposed. We have shown that our model is efficient and robust despite making no assumptions about the stochastic characteristics of the incoming jobs

## Keywords

Scheduling, Parallel computing, Resource management

## 1. INTRODUCTION

In the production world, supply chain optimization is of immense value. The effective use of the resources at hand, which means minimizing costs while maximizing throughput, is at the core of industrial research. Processes can be optimized by improving scheduling and process control and by reducing waste. While optimization problems typically involve assumptions about job arrival rates, type of jobs, and processing times, such assumptions are often non-realistic and thus generate inadequate results with a poor fit to the complex scenarios that are typical in reality.

In contrast to such modeling, in real life, realistic assumptions about these metrics cannot be made. For example, each new order that arrives at a production facility can vary from other orders in terms of order size and type, arrival time, and processing time. In such unstructured environments, the nature of the probability of each parameter cannot be accurately modeled. Moreover, at any given time in a production process, one cannot know whether a resource required to handle a particular job will be available. Likewise, once the resource is occupied, the stochastic nature of its availability also hinders modeling.

Optimization under such uncertainty is applicable to many production-like domains. In this work, we focus on the process of computing. As in traditional production, computing involves inputs (computing jobs) and outputs (computing results), which resemble the processes of conventional production. The 'conveyor' in our case is the computing infrastructure, and the goal is to optimize the processes or even just to ensure that they are efficiently managed by maintaining high and stable resource utilization, reducing idle computing power, and achieving overall fast and balanced computing results.

An integral component of modern industrial technology, the efficient and effective management of computing resources is of critical importance. Just as production processes and their optimums constituted the pinnacle of the industrial revolution, today efforts to optimize computing processes are at the forefront of the information age. But the rapid evolution of computing technology and computing performance metrics comes at the cost of an increased potential for resource waste.

Among the most ubiquitous manifestations of modern computing technology, distributed infrastructures – such as clouds, grids and clusters – have become a valid solution for data analysis in both the business and scientific computing worlds. Insofar as large-scale computing infrastructures are expensive, their utilization and efficiency are of great value. In multiuser and multi-machine computing environments, utilization and efficiency can be increased by sharing machines between multiple users. As modern data centers increasingly gravitate toward a reliance on shared clusters, an effective cluster resource management system, which is currently lacking, is of the utmost importance.

The significant improvements in the efficiency of computing technology notwithstanding, many limitations and challenges remain, particularly in the performance efficiency of both computing and communication. These limitations are augmented by the uncertainty of the streams of incoming computing jobs, which affects computing efficiency and brings additional challenges to bear on scheduling problems. Smooth and efficient computing operation therefore necessitates the design of robust and efficient resource management strategies.

The heterogeneous nature of computing environments is reflected in the variety of CPU, RAM, and I/O resources that are available. The application of these resources can be dedicated or shared, e.g., memory shared between multiple

cores on the same machine, which constrains the use of that memory by each core. Computing jobs submitted to these clusters may have markedly divergent resource demands, thus engendering a complex problem whose solution with resource allocation algorithms is challenging.

In this study, we explore resource utilization techniques given a defined set of variables, and we develop an adaptive model to address the problem at hand. The remainder of this paper is organized as follows. Section 2 establishes the background and includes a review of the related work in the field of resource allocation and task scheduling. Section 3 describes model analysis through simulation, and Section 4 contains the conclusions and suggested future research.

# 2. BACKGROUND REVIEW

In this section, we survey some of the main disciplines and concepts that are relevant to this study. Emphasis is given to resource scheduling and allocation of computing resources.

## 2.1 Scheduling

Scheduling allows the optimal allocation of resources among a given set of tasks to meet desired criteria. Formally, scheduling problems involve tasks that need to be scheduled on resources subject to system constraints and an objective to optimize some performance measures. The general aim is to build a schedule that specifies when and on which resource each task will be executed (Karger, et al., 2009).

In general, scheduling problems can be categorized as applicable to online or offline scheduling (Sgall, 1998). Scheduling decisions in the offline mode are made only after the complete set of jobs to be scheduled and other pertinent information, especially each job's runtime, are known. In contrast, scheduling decisions while online must be made – in lieu of the information that is already known before offline decisions are made – the instant that at least one job is ready to be executed. For real-time services, therefore, in which jobs must be processed ad-hoc and for which offline processing is not an option, the online scheduling schemes are highly relevant (Qureshi, et al., 2020). They are able to cope with both the lack of information about the future arrival of additional jobs and also the possibly unknown runtimes of the present jobs. In general, in both offline and online scheduling, the preemption of jobs is optional (Qureshi, et al., 2020) (Arndt, et al., 2000).

To efficiently coordinate resource sharing, achieve fairness and satisfy time constraints, cluster schedulers tend to exploit preemption, which is usually done by simply stopping the execution of the low priority jobs and restarting them later when resources are available. Thus, a non-sophisticated preemption policy causes significant resource waste and delays the response times of long running or low priority jobs (Li, et al., 2015).

Without preemption, however, conflicts in task completion can arise when tasks try to access a resource already occupied by another task. Offline, problems like this can be solved with a table-driven schedule while synchronization protocols can be used for runtime operation (Holenderski, et al., 2012). In the current work, however, preemption is not considered. Once a job has been assigned to a core on a machine, it remains there until it has finished. This is a reasonable assumption for a cluster of machines in which task migration is very costly, and it easily outweighs the possible improvements on the schedule.

In the cloud and cluster computing contexts, scheduling constitutes a major challenge. For cloud computing, scheduling methods are essential to improve throughput and utilization, to reduce costs and to provide the rapidest service times (Ibrahim, et al., 2021). Classic schedulers, with static internal behavior that shows no or very few alterations in resource structures, have been designed for batch processes in homogeneous environments. Nascent modern systems, like grid, cloud, fog, and edge computing (Qureshi, et al., 2020), in contrast, are defined by highly heterogeneous environments with variable structures (Berlińska, et al., 2011).

## 2.2 Resource management

To achieve effective job scheduling on a cluster of machines, resources must be properly managed. In a cloud environment, resource management is a hard problem due to the scale of modern data centers; the heterogeneity and inter-dependencies of the types of resources; load variability and unpredictability; and the scope of the sometimes conflicting objectives of the different actors in a cloud ecosystem. Consequently, both academia and industry have undertaken significant research in job scheduling (Jennings, et al., 2015).

Two of the most important concerns for users of shared environments are performance and fairness. Previous studies have shown that resource contention between users/jobs causes a trade-off between performance and fairness when considering an effective and efficient scheduling strategy (Niu, et al., 2015). While users are concerned with the performance of their applications, operators are usually more interested in efficient resource utilization (Kalra, et al., 2015).

Cloud computing operators try to achieve scaling capabilities by building large-scale datacenters and by sharing their resources between multiple users and workloads. Nevertheless, most cloud facilities operate at very low utilization, thus impinging on their cost effectiveness. Despite reservations that utilize up to 80% of the total capacity, aggregate CPU utilization is consistently below 20%. Typical memory use is higher (40-50%), but still less than the reserved capacity (Reiss, et al., 2012). This scenario reflects the same lazy approach of 'just throw some computing power at it', since the problem is not straightforward. Regardless of the above, to minimize operational risks, many cloud computing operators would overprovision their platforms. While operators could benefit by planning better and by utilizing smarter scheduling approaches to resource allocation and provisioning that will also reduce costs (Andreadis, et al., 2021).

Much of the literature on resource management deals with the allocation of resources to jobs, i.e., matchmaking between requirements and available resources. Cluster schedulers like YARN (Vavilapalli, et al., 2013) and Borg (Verma, et al., 2015) exploit a resource manager (RM), a logically centralized service that matches the resource needs of the different jobs with the available resources on worker machines. Every few seconds during their operation, machines communicate with the RM—worker machines report resource availability—and the RM allocates the tasks to the machines accordingly.

The main problem with this approach is in the location of the RM, namely, it is situated along the critical scheduling decision path. Additionally, resources can remain idle between the termination of a job and the next communication with the RM. This scenario can cause low cluster utilization, especially when a job consists of many small tasks (Rasley, et al., 2016). Moreover, it is in line with the results of the most recent

analyses released by Google of Borg cluster traces, wherein the average utilization of CPU and RAM is relatively low (Tirmazi, et al., 2020).

## 2.3 Resource allocation

Typical resource managers utilize resource allocation and assignment. Allocation refers to determining how much of each resource is used by a workload, i.e., number of servers, number of cores and amount of memory and bandwidth resources per server. Assignment refers to the selection of the specific resources needed to satisfy an allocation. The two greatest challenges of assignment comprise server heterogeneity and interference between co-located workloads, when hosts are shared to improve utilization (Delimitrou, et al., 2014).

Systems in which performance constraints need to be satisfied to fulfil system operations and end-user demands require dynamic resource allocation. This necessitates the development of efficient resource allocation strategies that take, as input, an application model, multi-core platform model and constraints, and that output real-time performance resource allocations or optimized resource allocations. Although several articles have been published about real-time allocation on multi/many-core systems and significant progress has been made in the field, myriad questions and research challenges remain open (Singh, et al., 2017). These challenges are exacerbated by the difficulty of identifying the means with which to accurately assess the status of the resources (for example, memory utilization on the system cores) during runtime. Since inappropriate resource management can lead to inefficiency and low runtime performance of the system, dynamic resource allocation methods are critical to improve utilization (Singh, et al., 2020).

The uncertainty inherent in resource allocation has yet to be adequately addressed in the scientific literature. Job runtimes constitute a major source of uncertainty due to the unpredictability of the workload, which can change dramatically. This unpredictability is due to the difficulty of accurately estimating the runtimes of submitted jobs (which typically exhibit a very low correlation with historical data), and of executing prediction correction, prediction fallback, etc. It is therefore important to design knowledge-free algorithms that consider effective alternatives to known optimization technologies that assume exact knowledge of the job parameters (Ramírez-Velarde, et al., 2017).

Another source of uncertainty is the heterogeneity of the submitted jobs to the clusters of machines. In such clusters, which can result in different demands on resources, some jobs like machine learning tasks are CPU-intensive while sorting task are memory-intensive. Furthermore, a task might have stringent resource requirements on multiple types of resources, e.g., memory and CPU, thus complicating the scheduling design even more (Grandl, et al., 2014).

## 2.4 Job scheduling

Cluster operators seek efficient utilization of the invested resources, while users are concerned with the overall experience and performance (Kalra, et al., 2015). Thus, effective job scheduling on the finite resources in computing clusters is of high importance to promote return on investment and meet user expectations. Consequently, the optimal scheduler would be capable of effective job distribution across the machines in a cluster. The scheduling decisions have to consider the mean execution time of the jobs, as well as efficient resources utilization to avoid resource waste, and optimize response times of the successfully completed jobs in order to improve the user experience (Soualhia, et al., 2017).

Computing resources are often not available instantly for a submitted job, therefore a queue must be maintained. There are two main approaches to implementing job scheduling queues – centralized and distributed. Centralized approaches implement centralized scheduler to queue the jobs, such centralized systems may be suffering from inherent feedback delays. Since worker nodes must frequently update the scheduler about their status. When most of the submitted jobs are short tasks, such clusters achieve suboptimal utilization, since the communication overhead becomes a bottleneck. In contrast, in distributed approaches, the queues are implemented on the worker-nodes where the submitted jobs are processed. Distributed approaches tend to achieve better cluster utilization. This comes with a cost, since distributed systems lack system-wide information, and thus may fail to schedule on the most suited processing resource and cause long job completion times. As a result, distributed clusters might have poor performance when the variability in job types is large (Rasley, et al., 2016).

The desired approach, therefore, must efficiently exploit the system's resources so the negative impacts of job performance issues on turnaround and on the waiting and response times are reduced as much as possible. System performance is primarily evaluated according to the average case. In the task scheduling literature, precedence is given to minimizing the variance in response time rather than its average. A system that exploits a sensible and expected response time may therefore be preferred to a system that might be faster on average but at the same time has a highly variable response time (Dave, et al., 2017).

It is not realistic to assume that the job flow will have low variance or be of a predictable stochastic nature. In addition, considerable scheduling challenges due to constraints-meeting objectives remain to be resolved. For example, *data locality*, i.e., to maximize system throughput, place computations near their input data; *memory constraints*, i.e., memory sharing by multiple cores limits the memory available for each core, etc. These and other hurdles imply the challenges entailed in the development of an adaptive model for load handling in machine clusters.

## 2.5 Adaptive approach

Adaptive approaches that are able to deal with a wide variety of workloads and grid properties have also been studied in (Ramírez-Alcaraz, et al., 2011; Hirales-Carbajal, et al., 2012). In all cases, job runtime estimation must be considered while performing resource allocation. The inaccuracy of such estimates, however, leaves room to significantly improve allocation strategy outcomes, especially in distributed environments (Ramírez-Alcaraz, et al., 2011).

A range of adaptive approaches, including some machine-learning methods, have been suggested in the literature as able to achieve high efficiency in resource allocation. For example, the probability of predicting queue waiting times can be modeled by using multi-class classification of similar jobs in history using dynamic k-Nearest Neighbors (k-NN) and Support Vector Machines (SVMs) (Kumar, et al., 2014). Others have proposed the efficient use of linear and quadratic regression models and decision trees (Kianpisheh, et al., 2012). However, all of the above methods rely on certain assumptions or adaptations to certain types of data, for example, a model adapted to heavily- and lightly-loaded processors (Albers,

1999), known runtimes, or job queue size. A static scheduling algorithm based on cost-reward optimization functions (truthful mechanism) is presented in (Grosu, et al., 2004). In (Ramirez-Velarde, et al., 2008), the authors applied self-similarity and heavy-tails to create scalability models for high performance clusters. Other adaptive approaches that are suggested use well-known scheduling platforms like YARN (Vavilapalli, et al., 2013) and provide automated parameter tuning techniques according to a predefined strategy and assumptions, which is a challenge for flow of jobs that lacks any statistical characterization (Herodotou, et al., 2020).

While most of the scheduling approaches in the extant research rely heavily on assumptions, in this work, we propose a different, robust and assumption-free approach for effective resource management and scheduling in the realistic and complex scenarios of distributed computing systems. In our simulated model, we assume a distributed system in which every machine has a shared resource, and we introduce a constraint on the amount of memory that is available on a machine. Therefore, in addition to scheduling, we also consider resource management algorithms. In the frame of this work, we sought to clarify how memory and computing cores can be fairly yet efficiently used.

# 3. SYSTEM MODEL

## 3.1 Problem definition

The main motivation of this work was to characterize a general model in the context of the classic production world. The proposed model may fit a variety of real-life scenarios in which it is not realistic to assume the exact stochastic nature of the process variables, i.e., the job arrival times and processing times.

The assumed highly irregular and unpredictable nature of the process variables renders known theoretical models irrelevant. We therefore propose a robust method based on real-time evaluation of the system state and dynamics, wherein the only inputs are the system's current state, and resource allocation decisions are derived accordingly.

As a case study, we use an important, real problem from the computing domain. We thoroughly analyze a relatively simple case and use the analysis results to draw conclusions about more complex scenarios. To that end, we constructed a topology of $N$ homogenous multicore machines – each machine with $S$ number of processing cores – as a logical cluster with a shared memory resource $R$. These machines executed a stream of jobs $J_j$ from multiple queues. Solving even this simplified scenario, however, is still highly challenging but warranted, as to the best of our knowledge, no similar efficient control model currently exists.

## 3.2 Model design

The general logical flow of jobs of our specific model is depicted in Fig. 1.

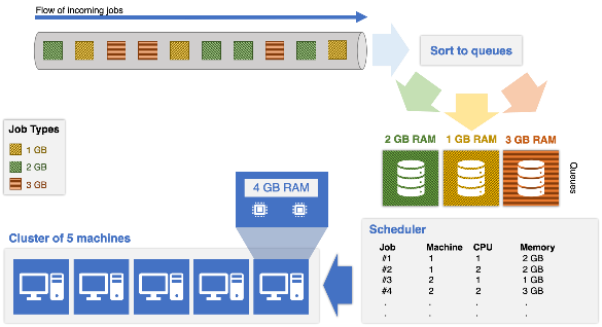


**Fig. 1: Logical flow of jobs**

Fig. 1 shows that the scheduler is tasked with matching job demands with the available core and memory resources. Managing both cores and memory, which constrain each other—i.e., two cores on the same machine have access to shared memory that amounts to 4 GB—is a complex task. We propose, as depicted in Fig. 2, to create an abstraction of cluster resources that represents three virtual clusters of resources and to manage them separately.

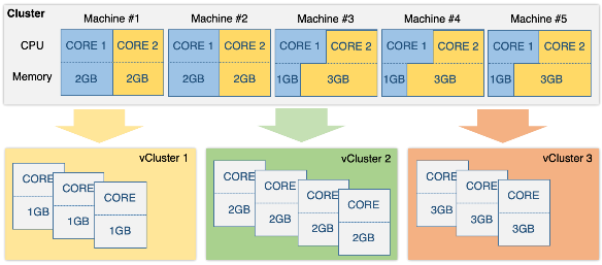


**Fig. 2: Abstraction of cluster resources**

Our proposed virtual clusters have an adaptive number of cores that already possess sufficient memory to receive the incoming jobs, which are assigned to three different queues. In the event that one of the queues becomes overloaded, we can simply provision more cores to the relevant pool of machines with appropriate memory allocation. This setup allows us to design a simple provisioning method, namely, we only need to define the rules about when to provision more cores – with appropriate memory – to a queue.

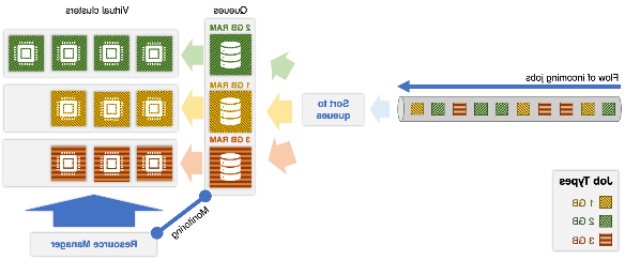

**Fig. 3: Flow of jobs with virtual clusters**

The abstraction shown in Fig. 3 allows us to develop resource provisioning strategies without the need to monitor both cores and memory. We were interested in two events in the system: queue change and end processing job events. Following are two pseudo-code algorithms that define how the two events are handled in the system.

**Algorithm 1:** Queue Change Event

**Input :** $Q_1$, $Q_2$, $Q_3$: job queues; *TR*: threshold; *vClusters*(3): virtual clusters;
**Output :** *vClusters*(3): virtual clusters.
1 $\theta \leftarrow$ `compareStrategy(`$Q_1$, $Q_2$, $Q_3$, *TR*`)`
2 **return** `provisionCores(`*vCluster*(θ)`)`

**Algorithm 2:** End Processing Job Event

**Input :** *J*: completed job; $Q_1$, $Q_2$, $Q_3$: job queues; *vClusters*(3): virtual clusters;
**Output:** *pCluster*: physical cluster.
1 **if** provisionMet(*pCluster* ,*vClusters*) **then**
2 **return** releaseCore(*J*)
3 **else**
4 **return** redesignateCore(*J*)

Algorithm 1 suggests that, based on the queue statistics and dynamics, we decide whether to re-provision the cores in the system adjust it to the current flow of jobs. Algorithm 2 suggests that when a job is completed, we first check whether a re-provision of the core is needed and then release the core accordingly, thus enforcing the strategy of resource provisioning given by Algorithm 1.

## 3.3 Model simulation

The model was implemented with MATLAB® and Simulink® version R2018a, and the SimEvents® library was used. The MATLAB file for the model in this work is available for download at

https://drive.google.com/file/d/1_udAxvsBkVnb15ZbdB6ed6pvgIoB_vaq/view?usp=sharing

## 3.4 Incoming job creation in simulation

No assumptions were made about the system's incoming jobs. To manage the job creation process in the simulation, we had to define a unique creation process (Fig. 4).

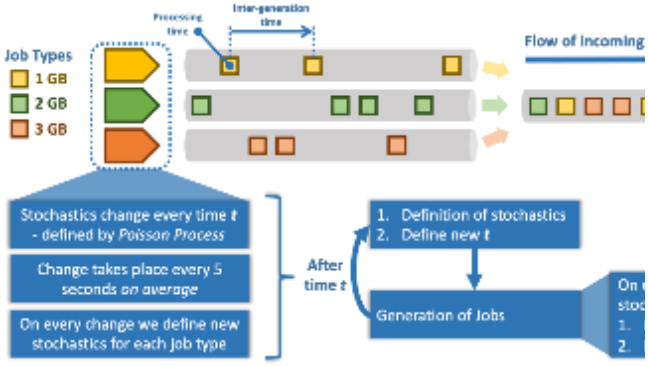


**Fig. 4: Job creation process in the simulation**

We designed the stream of incoming jobs such that the system will be loaded to approximately 80% capacity, i.e., on a system comprising 10 cores, an average of 8 cores will be occupied. The rationale for this approach was dictated by the shared memory constraints, namely, some cores will not be able to accept computing jobs because they do not have enough available memory when that memory is being used by another core. If we drive a stream of jobs that utilizes the entire cluster, 100% of the time, the incoming jobs will begin to accumulate in the job queues.

## 3.5 Provisioning strategy

We exploited a simple yet powerful strategy to adapt the number of cores available for each queue. We equalized the proportion of queue *occupancy* of each queue with the proportion of cores provisioned to the queue; in the event that the proportional occupancy of a given queue is larger than the proportion of cores provisioned to that queue by some threshold *TR*, then we allocate more cores to that queue. Occupancy, which is calculated by multiplying the total waiting time in the queue by the queue length, allows one to estimate the extent to which the resources for a queue are 'occupied'.

For example, re-provision decisions that are made based only on queue length may cause a 'burst' of jobs of the same type, thus lengthening the queue (but the processing times of these jobs may be very short). On the other hand, making the same decisions based only on waiting time may cause jobs that are already in a queue to take long times to fully process (yet they may be few in number). This observation indicates that considering both the waiting times and the queue lengths will confer on the process the benefits of both approaches simultaneously.

## 3.6 Model performance

To evaluate model performance, we collected certain statistics, such as measures of core and memory utilization.

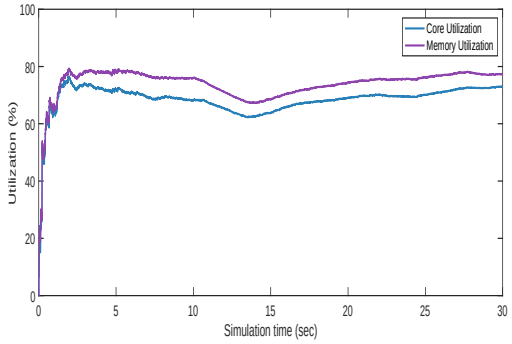


**Fig. 5: Utilization of cores and memory during simulation**

The performance metrics depicted in Fig. 5 show that the many changes in job stream stochastics, model performance exhibits approximately 70-75% utilization in comparison with a stable stochastics stream of jobs (i.e., the arrival rate for the Poisson process remains the same over the course of the entire simulation), under which utilization is 75% under a similar, average load.

Fig. 6 shows that the system receives a variable flow of jobs and that at simulation time 5 s, a job peak requiring 3 GB of memory begins. Since the system was designed to be adaptive, however, we expect it to reallocate computing resources to the third (3 GB) queue to handle the unpredicted peak.

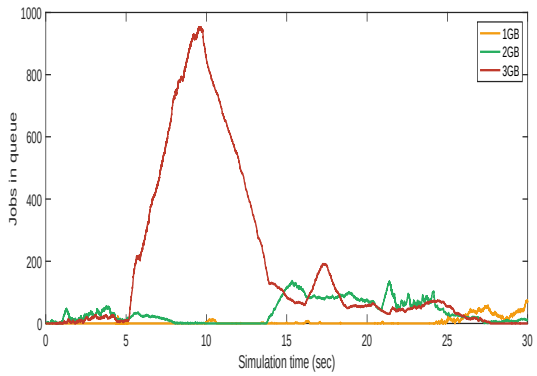


**Fig. 6: Number of jobs queued during simulation**

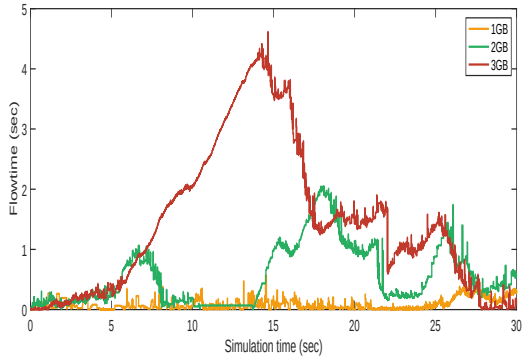


**Fig. 7: Flowtime of jobs during simulation**

The duration of time required for a job to *flow* through the system until its completion is the flowtime metric (i.e., the elapsed time from job entrance to the system until the end of job processing). The flowtime metric differs from the waiting time metric in that the former refers only to the processing time. Figure 7 shows that, despite the queue lengths, the flowtime was comparably short, an outcome that is due to the reallocation of resources to the queues with high job occupancies. Moreover, the graph in Fig. 7 also indicates that the system managed to reduce the flowtime after the peak in incoming jobs.

## 4. CONCLUSIONS

In this work, we addressed a scheduling problem in a cluster of computers, each with two cores, with a total shared memory of 4 GB. Highly realistic and robust, this model makes no assumptions about job arrivals or job processing times. In addition, the resources were represented as designated pools of computing resources with pre-assigned memory capacities of 1, 2 or 3 GB, thus facilitating core provisioning and allowing us to focus on the scheduling strategy. We showed that the model is adaptive and adequately robust.

This model can serve as a framework for the development of more complex scenarios. Though the model in this work assumed only three types of jobs with respect to memory requirements, in reality, memory requirements are typically of a continuous nature. Future works should therefore explore more sophisticated representations of the core and memory allocation “duo”. To that end, future research should address provision strategies, especially the method that is used to determine changes in provisioning. For example, perhaps the use of a dynamic threshold that changes in conjunction with system statistics could enable more advanced and adequate switching between system states.

In addition, this work did not consider the preemption of processes, i.e., halting jobs during processing and switching computing resources to another job in the queue. Moreover, it did not address cases such as the simultaneous scheduling of multiple jobs on the same core, which could be managed by switching between jobs, thus allocating resources not only at the computing core level, but explicitly assigning a computing time slot for a job. Both of these scenarios can be integrated in this model.

This study, to the best of our knowledge, is the first to drop any assumptions on process stochastic nature of job arrivals and job processing times and to consider the constraints imposed by shared memory in multicore machines assembled in a cluster.

## 5. REFERENCES

[1] D. Karger, C. Stein and J. Wein, 2009, "Scheduling Algorithms," in Scheduling Algorithms. Algorithms and Theory of Computation Handbook: special topics and techniques, vol. 2, Chapman and Hall/CRC.

[2] J. Sgall, 1998,"On-line Scheduling," in Online Algorithms, Springer, Berlin, Heidelberg, pp. 196-231.

[3] M. S. Qureshi, M. B. Qureshi, M. Fayaz, W. K. Mashwani, S. B. Belhaouari, S. Hassan and A. Shah, 2020, "A comparative analysis of resource allocation schemes for real-time services in high-performance computing systems," Journal of Distributed Sensor Networks, vol. 16, no. no. 8.

[4] O. Arndt, B. Freisleben, T. Kielmann and F. Thilo, 2000, "A comparative study of online scheduling algorithms for networks of workstations," Cluster computing, vol. 3, no. 2, pp. 95-112.

[5] J. Li, C. Pu, Y. Chen, V. Talwar and D. Milojicic, 2015, "Improving Preemptive Scheduling with Application-Transparent Checkpointing in Shared Clusters," in Middleware '15 Proceedings of the 16th Annual Middleware Conference, Vancouver, BC, Canada.

[6] M. Holenderski, R. J. Bril and J. J. Lukkien, 2012, "Parallel-Task Scheduling on Multiple Resources," in Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on Real-Time Systems.

[7] I. M. Ibrahim, S. R. M. Zeebaree, M. A. M.Sadeeq, A. H. Radie, H. M. Shukur, H. M. Yasin, K. Jacksi and Z. N. Rashid, 2021, "Task scheduling algorithms in cloud computing: A review," Turkish Journal of Computer and Mathematics Education, vol. 12, no. 4.

[8] J. Berlińska and M. Drozdowski, 2011, "Scheduling divisible MapReduce computations Author links open overlay panel," Journal of Parallel and Distributed Computing, vol. 71, no. 3, pp. 450-459.

[9] B. Jennings and R. Stadler, 2015, "Resource Management in Clouds: Survey and Research Challenges," Journal of Network and Systems Management, vol. 23, no. 3, p. 567–619.

[10] Z. Niu, S. Tang and B. He, 2015, "Gemini: An Adaptive Performance-Fairness Scheduler for Data-Intensive Cluster Computing," in 2015 IEEE 7th International

Conference on Cloud Computing Technology and Science (CloudCom), Vancouver, BC, Canada.

[11] M. Kalra and S. Singh, 2015, "A review of metaheuristic scheduling techniques in cloud computing," Egyptian informatics journal, vol. 16, no. 3, pp. 275-295.

[12] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz and M. A. Kozuch, 2012, "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis," in SoCC '12 Proceedings of the Third ACM Symposium on Cloud Computing, New York.

[13] G. Andreadis, F. Mastenbroek, V. v. Beek and A. Iosup, 2021, "Capelin: Data-Driven Compute Capacity Procurement for Cloud Datacenters using Portfolios of Scenarios," IEEE Transactions on Parallel and Distributed Systems.

[14] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed and Baldeschwiele, 2013, "Apache Hadoop YARN: yet another resource negotiator," in SOCC '13 Proceedings of the 4th annual Symposium on Cloud Computing, Santa Clara, California.

[15] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune and J. Wilkes, 2015, "Large-scale cluster management at Google with Borg," in EuroSys '15 Proceedings of the Tenth European Conference on Computer Systems, Bordeaux, France.

[16] J. Rasley, K. Karanasos, S. Kandula, R. Fonseca, M. Vojnovic and S. Rao, 2016, "Efficient Queue Management for Cluster Scheduling," in EuroSys '16 Proceedings of the Eleventh European Conference on Computer Systems, ondon, United Kingdom.

[17] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter and J. Wilkes, 2020, "Borg: the next generation," in Proceedings of the Fifteenth European Conference on Computer Systems.

[18] C. Delimitrou and C. Kozyrakis, 2014, "Quasar: Resource-Efficient and QoS-Aware Cluster Management," in ASPLOS '14 Proceedings of the 19th international conference on Architectural support for programming languages and operating systems.

[19] A. K. Singh, P. Dziurzanski, H. R. Mendis and L. S. Indrusiak, 2017, "A Survey and Comparative Study of Hard and Soft Real-Time Dynamic Resource Allocation Strategies for Multi-/Many-Core Systems," ACM Computing Surveys (CSUR), vol. 50, no. 2.

[20] H. Singh, A. Bhasin, P. R. Kaveri and V. Chavan, 2020,"Cloud Resource Management: Comparative Analysis and Research Issues.," INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH, vol. 9, no. 06, pp. 96-113.

[21] R. Ramírez-Velarde, A. Tchernykh, C. Barba-Jimenez, A. Hirales-Carbajal and J. Nolazco-Flores, 2017, "Adaptive Resource Allocation with Job Runtime Uncertainty," Journal of Grid Computing, vol. 15, no. 4, pp. 415-434.

[22] R. Grandl, G. Ananthanarayanan1, S. Kandula, S. Rao and A. Akella, 2014, "Multi-Resource Packing for Cluster Schedulers," in SIGCOMM '14 Proceedings of the 2014 ACM conference on SIGCOMM, New York.

[23] M. Soualhia, F. Khomh and S. Tahar, 2017, "Task Scheduling in Big Data Platforms: A Systematic Literature Review," The Journal of Systems and Software, vol. 134, pp. 170-189.

[24] B. Dave, S. Yadav and M. Mathuria, 2017, "Customary Methods for CPU Scheduling : A Review," International Journal of Scientific Research in Science and Technology, vol. 3, no. 8.

[25] J. M. Ramírez-Alcaraz, A. Tchernykh, R. Yahyapour, U. Schwiegelshohn, A. Quezada-Pina, J. L. González-García and A. Hirales-Carbajal, 2011, "Job Allocation Strategies with User Run Time Estimates for Online Scheduling in Hierarchical Grids," Journal of Grid Computing, vol. 9, no. 1, pp. 95-116.

[26] A. Hirales-Carbajal, Tchernykh, A., Yahyapour, R., González-García, J. L., Röblitz, T. and Ramírez-Alcaraz, J. M., 2012, "Multiple workflow scheduling strategies with user run time estimates on a grid," Journal of Grid Computing, 10(2), pp. 325-346.

[27] R. Kumar and S. Vadhiyar, 2014, "Prediction of queue waiting times for metascheduling on parallel batch systems," in Workshop on Job Scheduling Strategies for Parallel Processing.

[28] S. Kianpisheh, S. Jalili and N. M. Charkari, 2012, "Predicting Job Wait Time in Grid Environment by Applying Machine Learning Methods on Historical Information," International Journal of Grid and Distributed Computing, vol. 5, no. 3, pp. 11-22.

[29] S. Albers, 1999, "Better Bounds for Online Scheduling," SIAM Journal on Computing, vol. 29, no. 2, pp. 459-473.

[30] D. Grosu and A. T. Chronopoulos, 2004, "Algorithmic mechanism design for load balancing in distributed systems," IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 34, no. 1, pp. 77-84.

[31] R. Ramirez-Velarde, C. Vargas, G. Castañon and L. Martinez-Elizalde, 2008, "Self-similarity and Multidimensionality: Tools for Performance Modelling of Distributed Infrastructure," in On the Move to Meaningful Internet Systems: OTM 2008.

[32] H. Herodotou, Y. Chen and J. Lu, 2020, "A survey on automatic parameter tuning for big data processing systems," ACM Computing Surveys (CSUR), vol. 53, no. no. 2, pp. 1-37.