**Mathematics and Computer Science**
2021; 6(3): 49-58
http://www.sciencepublishinggroup.com/j/mcs
doi: 10.11648/j.mcs.20210603.12
ISSN: 2575-6036 (Print); ISSN: 2575-6028 (Online)

# Enhancing Parallel Scheduling of Grid Jobs in a Multicored Environment

**Goodhead Tomvie Abraham[1], Evans Fiebibiseighe Osaisai[2], Abalaba Ineyekineye[2]**

[1]Computer Science Department, Niger Delta University, Yenagoa, Nigeria

[2]Mathematics Department, Niger Delta University, Yenagoa, Nigeria

## Email address:

at.goodhead@gmail.com (G. T. Abraham), fevansosaisai@gmail.com (E. F. Osaisai), ineyeabalaba@yahoo.com (A. Ineyekineye)

## To cite this article:

Goodhead Tomvie Abraham, Evans Fiebibiseighe Osaisai, Abalaba Ineyekineye. Enhancing Parallel Scheduling of Grid Jobs in a Multicored Environment. *Mathematics and Computer Science*. Vol. 6, No. 3, 2021, pp. 49-58. doi: 10.11648/j.mcs.20210603.12

**Received**: May 17, 2021; **Accepted**: June 9, 2021; **Published**: June 21, 2021

**Abstract:** The computing Grid has emerged as a platform to solve the complex and ever-increasing processing need of man and advances in computing technology have birthed the multicore era aimed for high throughput and efficient parallel computing. However, most systems still rely on the underlying hardware for parallelism despite the hard evidence that sequential algorithms do not optimally exploit parallel systems. This research seeks to harness the benefits of multicore systems using job and machine grouping methods to enhance parallelism in the scheduling of Grid jobs. The paper presents the result of two separate experiments on a method that parallelize scheduling algorithm on two multicore platforms. An arbitrary method was employed to group machines; a summation of the total processing power of machines in each group was made. To ensure load balancing, jobs were allocated to machine groups based on the ratio of the total processing power of the machines in each group. The MinMin Grid scheduling algorithm was implemented independently within the groups using a range of threads varied in powers of two. Also, the numbers of groups were varied between 2, 4, and 8. The same experiment was executed on a single processor computer; a duocore machine and a quadcore machine. A performance improvement of 16% to 85% was recorded by the group method against the best ordinary MinMin results and an improvement of 50% to 84% was recorded by the group method against the ordinary MinMin on corresponding machines. We prove that an increase in the number of groups results in improved performance on corresponding machines (approximately 2 times using 2 groups, approximately 3 times using four groups, and approximately 6 times using 8 groups). And most importantly, we established that as the number of processors increases, the grouping method makes more significant improvements over the ordinary MinMin scheduling algorithm executed on the multicore systems.

**Keywords:** Multicore-environment, Parallelism, Multi-scheduling, Machine Grouping, Job Grouping, Scheduling

## 1. Introduction

The advent of Grid computing has been acclaimed as the paradigm to solve the ever-increasing computing need of an ever-demanding world while multicore systems have been heralded as the major architecture choice for modern computing platforms - this is anticipated to remain so for long [1, 2] and [3]. However, it has been shown that sequential algorithms do not gain much from parallel systems if the algorithm is not Parallelized [4]. Current Grid scheduling algorithms are mostly sequential and do not exploit the inherent benefits in the underlying multicore systems, while most others focus on scheduling parallel jobs rather than scheduling jobs in parallel. Scheduling of Grid jobs without exploiting the underlying multicore hardware in this era of multicore systems poses a negative trend for the growth and purpose of the Grid. The method presented in this work provides a general means of Parallelization of sequential algorithms. The method uses several independent groups to enable several scheduling instances within the group (multi-scheduling), this greatly enhances parallelism.

The remainder of the paper is organized as follows: the next section discusses related literature. Section 3 discusses the proposed method and presents results and analysis. Section 4 makes recommendation and section 5 discusses conclusion and future thoughts.

# 2. Related Work

The NP-completeness of heterogeneous systems requires heuristics to ease the problem-solving. Exploiting a method that enhances the efficiency of multicore systems for the overall benefit of grid scheduling will be beneficial to the growth of the Grid. Advances in computer hardware technology are aimed at parallelizing processing, high throughput, performance improvement, speedup, and efficiency [5]. However, most Grid scheduling algorithms remain sequential; such algorithms 'eat up' the gains in hardware technology', impede performance, and do not guarantee maximum speedup [6, 7] and [4]. The multicore era, therefore, requires a parallel approach to programming [8-10]. However, parallelizing sequential algorithms are quite complex and not completely guaranteed. Therefore, a method that enhances parallelism in the execution of all algorithms is sought; grouping jobs and machines before simultaneously scheduling independent groups (multischeduling) enhances parallelism on the multicore systems and increases throughput [11].

This work uses job and machine groups to exploit parallelism on multicore systems.

### *2.1. Parallelism and Multicores*

Serial computing is replete with drawbacks [12], multicore technology was, therefore. the direction for future computing interest [13] and [14]. However, major advances in computing technology come with a paradigm shift in programming [15]; based on calls by many researchers to embrace parallelism; a better programming method that enhances the performance of multicore systems should therefore be of interest to researchers because the potentials of multicore systems are largely being under-utilized on current systems [7, 16, 17].

In this regard, several researchers have experimented with the execution of algorithms on parallel systems: an experiment was conducted on clusters of CPU-GPU to speed up queries from high dimensional holistic data that are being updated constantly, the experiment attained efficiency in handling queries in [18]. A method that exploited parallel algorithms to scale multicore systems and obtained optimal results was presented in [19].

A method that used a code transformation to create data parallelism latent in parallel applications to enhance resource utilization and speedup on chips to facilitate parallelism on the hardware was exploited in [20]. While a method that efficiently executed (Precedence-constrained Task Graphs) PTGs on a distributed system with heterogeneous processing elements connected through a set of shared heterogeneous buses was proposed in [21].

A novel framework that automatically, efficiently, and recursively computes the divide and conquer algorithms for a set of dynamic programming problems in multicore systems was exploited in [22]. In the same vein, optimization in energy consumption and improvement in schedule length was achieved by employing a parallelizing scheduling method to find a solution to the duplication-based scheduling problem in [23]. Also, an experiment on parallel programming paradigms like OpenMP on CPU cluster and CUDA on GPU cluster using (Breadth-First Search) BFS and (Depth-First Search) DFS graph algorithm that attained a speedup of 187 to 240 by CUDA on GPU over the OpenMP on CPU clusters was executed in [24]. They however noted that the CPU clusters underperformed due to communication overheads and idle time.

The gains of parallel computing systems have been exploited by researchers in other fields as well: In bioinformatics; parallel programming technology was employed for genome sequence processing in [25]. In electronics, a parallel computation method to curb the bottleneck incurred by serial computation of CRC was exploited in [26]. The method efficiently increased speed-up in the computation of (cyclic redundancy check) CRC computation on CPU and Field Programmable Gate Array (FPGA). And in geospatial statistics; parallelism was exploited on a parallel hardware architecture in the computation of environmental data in [27] and [28].

Though these researches all aimed at attaining throughput, speed-up, and optimize processing on parallel architecture; no special method was employed to enhance parallelism. However, optimal exploitation of parallel architecture requires a fundamental approach to the way programming is done [29, 30, 8, 10, 7]. This requirement necessitated the group-based approach undertaken in this work.

### *2.2. Grouping of Jobs Before Scheduling*

Grouping of jobs before scheduling has been used extensively by researchers to achieve different aims. A method that grouped fine-grained jobs to form coarse-grained jobs before scheduling to improve response time was exploited in [31]. A method that grouped jobs before transmitting to Grid resources for computation was exploited in [32]. While another method that exploited grouping to reduce the (communication computation ratio) CCR before the schedule was presented in [33]. Also, a method that divides the base relation into several independent sub-cubes to reduced exponentiality and efficiently performed queries was exploited [18] – here, the sub-cubes are likened to groups.

In these works; parallelism dependent on the underlying hardware; the group was used to attain optimality or efficiency but not really in aiding parallelism.

This research employs grouping as platforms to parallelize the scheduling of grid jobs on multicore systems. The groups enable the threads to execute independently in multi-scheduling of jobs within the groups - taking advantage of the multicores.

In line with this research, significant improvement was achieved in [11] using job and machine groups before scheduling. However, the effect of increasing the number of groups could not be investigated further. As a result, a dynamic means of grouping grid jobs and grid machines for efficient multi-scheduling was proposed in [34]. The method recorded significant improvement with increasing groups.

Hence, various strategies to group machines and Grid jobs before scheduling on an HPC system were experimented with in [35] and significant improvement was recorded by all methods as the number of groups increases. However, on the HPC system, the number of CPUs is constant and the effect of varying the number of processors on the grouping method was recommended for investigation. A random machine grouping method and size-proportional-to-speed method of job grouping were proposed in [36] and a performance improvement of 16% to 71% on a duocore machine was achieved. The researchers noted that investigating further on a system with more cores was necessary to know the effect on the method. So, the same work was extended on a quadcore system in [37] and an improvement of 50% to 86%. However, the trend of improvement on both platforms needed to be examined. This paper combines the results and added more analysis of the separate experiments carried out in [36] and [37].

# 3. Parallel Scheduling of Grid Jobs in a 'Multicored' Environment

This research aims at exploring the parallelism inherent in multicore systems by using a method of job and machine grouping. An arbitrary method was employed to group machines; a summation of the total processing power of machines in each group was made. To achieve load balancing, jobs were then allocated to the machine groups based on the ratio of the total processing power of the machines in each group.

A range of one to eight threads in powers of two (1 to 8 step $2^n n \in [0,1,2,3]$) was used in varied experiments. Also, the number of groups was varied between 2, 4 and 8. The experiment was executed on a single processor system, a duocore system and a quadcore system.

The MinMin algorithm [38] was chosen as a benchmark in this work because it has been adopted by several other researchers for benchmarking [39-48].

### *3.1. Job and Machine Grouping*

The algorithms for grouping jobs and machines used in this experiment have been presented in [36, 37].

### *3.2. Experimental Design*

This work is based on the series of experiments carried out in [36] and [37].

### *3.3. Results and Data Analysis*

This section discusses results from all the experiments from the two previous works referred in section 3. In the discussion, 1Thrds, 2Thrds, 4Thrds or 8Thrds refer to the number of threads used in the experiment. SingleCPU or SingleProc, 1CPU or 1Proc refer to result obtained on the single processor machine; duocore refers to results from the duocore machine while Quadcore refers to results obtained on the quadcore machine. Also, 2Grp, refers to two groups, 4Grp refers to four groups and 8Grp refers to eight groups. The proposed method Size-Proportional-to-Speed and random method abbreviated as SpdRnd. A combination of group numbers, number of threads, and machines are used to refer to a specific result. For instance, 4ThrdsDuocore4Grps represent results obtained using 4 threads and 4 groups on the duocore machine.

### *3.4. Performance of the MinMin Algorithm on the Three Computing Platforms*

This section discusses the result of the ordinary MinMin (MinMin without the group method). Table 1 shows the results and performance of the ordinary MinMin on the three different computer platforms (single processor machine, duocore machine, and quadcore machine) and the computed performance improvements. The Single processor system took a total of 1235755 ms to schedule the range of jobs. The duocore system used a total of 220726 ms to schedule the range of jobs while the quadcore machine used 136818ms to schedule the same range of jobs (from 1000 to 10,000 step 1000).

Using the ordinary MinMin, there was a performance improvement of 5.6 times or 82% by the Duocore machine over the Single processor machine. The quadcore machine improved performance 9 Times or 88% over the single processor machine. And the Quadcore machine improved performance 1.6 times or 38% over the duocore machine.

These results show that the MinMin algorithm is scalable and gained immensely from the multicore system's underlying parallelism. However, the question is are these improvements recorded by the MinMin on the duocore and quadcore machines enough not to try a different method that enhances parallelism on the multicore systems? Analysis of results in sections 3.5, 3.6, and 3.7 between the performance of the MinMin and the group method on the multicore systems will provide answers.

***Table 1.*** *Scheduling result of MinMin algorithm on the three machines.*

| Jobs Limit | MinMin2Thrds (SingleCPU) | MinMin2Thrds (DuoCore) | MinMin4Thrds (QuadCore) |
|---|---|---|---|
| 1000 | 4500 | 690 | 399 |
| 2000 | 17672 | 2800 | 1795 |
| 3000 | 40250 | 6410 | 4222 |
| 4000 | 66922 | 10674 | 7188 |
| 5000 | 90407 | 26432 | 9994 |
| 6000 | 116922 | 19275 | 12700 |
| 7000 | 154781 | 24914 | 16690 |
| 8000 | 196768 | 32446 | 21719 |

| Jobs Limit | MinMin2Thrds (SingleCPU) | MinMin2Thrds (DuoCore) | MinMin4Thrds (QuadCore) |
|---|---|---|---|
| 9000 | 244679 | 44057 | 27989 |
| 10000 | 302854 | 53028 | 34122 |
| Total | 1235755 | 220726 | 136818 |
| Average | 123575.5 | 22072.6 | 13681.8 |
| Improvement over Single processor system (X) | | 5.60 | 9.03 |
| Improvement over Single processor system (%) | | 82.14 | 88.93 |
| Improvement over Duocore system (X) | | | 1.61 |
| Improvement over Duocore system (%) | | | 38.015 |

### *3.5. Analysis of Results on Corresponding Machines*

This section compares the result between the MinMin and the group method on corresponding machines.

Table 2 shows the results, computed total, Average, and performance improvement in multiples (X) and percentage (%) between the MinMin and the group methods on corresponding machines. This is also shown in Figure 1 while Figure 2 shows the improvement by the group method over the MinMin on the single processor machine, duocore machine, and the quadcore machine.

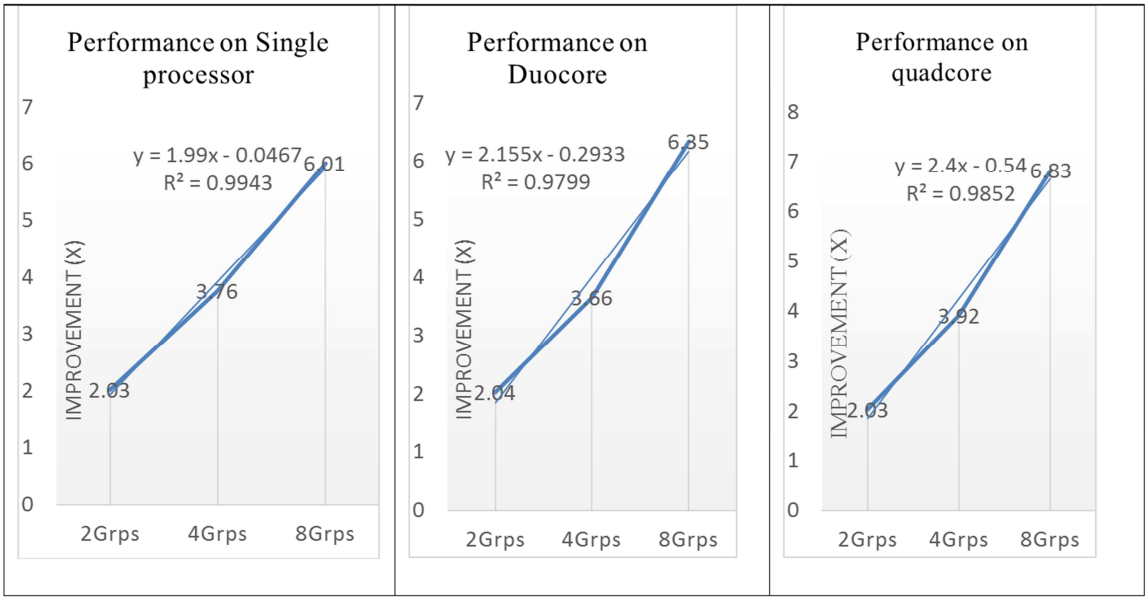


***Figure 1.*** *Performance improvement by group method on corresponding machines.*

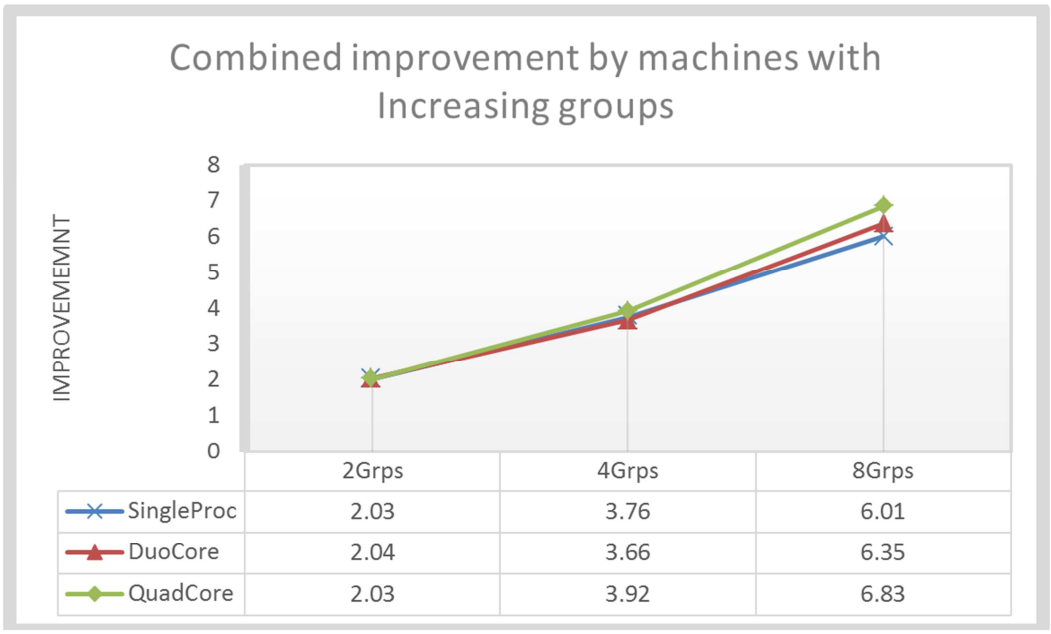


***Figure 2.*** *Combined performance improvement by group method on corresponding machines.*

On the single processor machine, the group method performed 2.03 times, 3.76 times, and 6.01 times better than the MinMin when executed with 2, 4, and 8 groups respectively. This represents 50%, 73%, and 83%. This result and computed improvement indicate that the group method performed better than the ordinary MinMin even on the single processor system.

The linear trendline on the improvement yielded the equation Y = 1.99x – 0.0467 and the R-Squared value of the trendline is 0.9943.

On the duocore machine, the performance was 2.04 times, 3.66 times and 6.35 times better than the MinMin which represents 50%, 72% and 84% when executed with 2, 4 and 8 groups respectively. The linear equation on the duocore performance trendline Y = 2.155x – 0.2933 with the R-squared value of 0.9799.

On the quadcore machine, the group method performed 2.03 times, 3.92 times and 6.88 times better than the MinMin algorithm. This represents 50%, 74% and 85% when executed with 2, 4 and 8 groups respectively. The trendline on the improvement yielded equation Y = 2.4x – 0.54 and the R-Squared value of the trendline is 0.9852.

The correlation of results and improvements between the group method and MinMin on corresponding machines is (0.99) which indicates that the group methods results are strongly reliable and acceptable.

The linear equations indicate that performance improvement Y depends on the number of groups (x) and increases with an increase in the number of groups. While the R-Squared value of 0.9x on the trendlines indicates a good fit, it also indicates the reliability and repeatability of the results.

Even though the MinMin is also scalable, the result indicates that the group method improves the performance of the parallel architectures (multicore systems) by an approximate factor of 2 when using two groups, an approximate factor of 3 when using four groups and an approximate factor of 6 when using 8 groups.

Figure 2 and the linear equations generated from the improvement trendline from the three computing platforms indicate that there is a general performance improvement with increasing cores.

***Table 2.*** *Analysis of corresponding results and performance.*

| | **Single Processor** | | | | **Duocore** | | | | **Quadcore** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **MinMin** | **2Grps** | **4Grps** | **8Grps** | **MinMin** | **2Grps** | **4Grps** | **8Grps** | **MinMin** | **2Grps** | **4Grps** | **8Grps** |
| 1000 | 4500 | 2360 | 1094 | 594 | 690 | 375 | 319 | 95 | 403 | 189 | 111 | 78 |
| 2000 | 17672 | 7907 | 3876 | 2031 | 2800 | 1360 | 617 | 331 | 1775 | 759 | 337 | 196 |
| 3000 | 40250 | 17031 | 9063 | 4360 | 6410 | 2628 | 1334 | 609 | 4203 | 1543 | 648 | 396 |
| 4000 | 66922 | 28468 | 13656 | 9359 | 10674 | 4665 | 2138 | 1575 | 7431 | 2570 | 1297 | 671 |
| 5000 | 90407 | 40453 | 22672 | 12985 | 26432 | 7680 | 3832 | 2323 | 9903 | 4339 | 1928 | 1178 |
| 6000 | 116922 | 59781 | 32578 | 19062 | 19275 | 9607 | 5481 | 2927 | 12984 | 6433 | 3089 | 1568 |
| 7000 | 154781 | 79499 | 42047 | 26234 | 24914 | 14322 | 7888 | 4864 | 16872 | 9079 | 4315 | 2361 |
| 8000 | 196768 | 94781 | 54172 | 34453 | 32446 | 17156 | 9967 | 5672 | 21956 | 11261 | 5758 | 2855 |
| 9000 | 244679 | 119609 | 67438 | 41250 | 44057 | 22976 | 12796 | 7273 | 27778 | 14340 | 7082 | 4506 |
| 10000 | 302854 | 157970 | 82423 | 55282 | 53028 | 27406 | 15921 | 9080 | 34570 | 17385 | 10593 | 6379 |
| Total | 1235755 | 607859 | 329019 | 205610 | 220726 | 108175 | 60293 | 34749 | 137875 | 67898 | 35158 | 20188 |
| Avg | 123576 | 60785.9 | 32902 | 20561 | 22073 | 10817.5 | 6029 | 3474.9 | 13787.5 | 6789.8 | 3515.8 | 2018.8 |
| Improvement (X) | | 2.03 | 3.7 | 6.01 | (X) | 2.04 | 3.66 | 6.35 | (X) | 2.03 | 3.92 | 6.83 |
| Improvement (%) | | 50 | 73 | 83 | (%) | 50 | 72 | 84 | (%) | 50 | 74 | 85 |

### *3.6. Comparison of Result on the Three Computing Platforms Against the Best MinMin Result*

This section compares the best ordinary MinMin result (obtained on the quadcore) against the group method result from other platforms. On the three computing platforms, the ordinary MinMin result from the quadcore machine was the best. Hence, the ordinary MinMin result yielded by execution on the quadcore system was used.

Table 3 shows the result and computed total scheduling time of the ordinary MinMin algorithm executed on the quadcore compared against results of the group method from the three platforms. It also shows the computed total and average time used in scheduling the range of jobs. While Table 4 and Table 5 shows the computed performance improvement between the machines in multiples (X) and in percentage (%) respectively.

From Table 4 and Table 5, the MinMin algorithm executed on a quadcore system performed better than the group method executed on the single processor system by 4.40 times, 2.37 times and 1.51 times (or 77%, 57% and 33%) when using two, four and eight groups respectively. This is shown in the first part of Figure 3 labeled A to B (and Figure 4 labeled Single processor). As the number of groups increases, it can be seen that the graph was falling from left to right – this indicates that the group method was able to increase performance as the number of groups increases – though not enough to match the performance improvement of the MinMin on the quadcore machine.

This result indicates that the MinMin algorithm is scalable and benefitted from parallelism on the quadcore while the group method executed on the single processor system underperformed due to the absence of parallelism which the method requires but could not be supported by the single processor system. The group method targets multicore systems, so it is not suited for single processor systems.

The second part of the graph in Figure 3 and Figure 4 (labelled C to D or duocore) shows the performance of the duocore system using the group method over the ordinary MinMin executed on the quadcore. The group method on the

duocore performed better than the MinMin by 1.21 times, 2.04 times and 3.56 times (or 16%, 49% and 71%) when using two, four and eight groups respectively for scheduling. This caused the aggregate graph to rise at a point beginning from duocore 2 groups and continued to rise to duocore 8 groups.

The group method exploited the parallelism inherent on the Duocore system and this resulted in the improved performance over the ordinary MinMin algorithm executed on quadcore.

Points E to F in Figure 3 (shown as quadcore in Figure 4) shows the performance of the quadcore system over the ordinary MinMin as the group increases from 2 to 8 groups. The quadcore machine recorded performance improvement of 2.03 times, 3.98 times and 7.10 times (or 50%, 74% and 85%) over the MinMin when using 2, 4 and 8 groups respectively.

This analysis indicates that performance improvement is attained both by an increase in the number of groups and an increase in the number of cores.

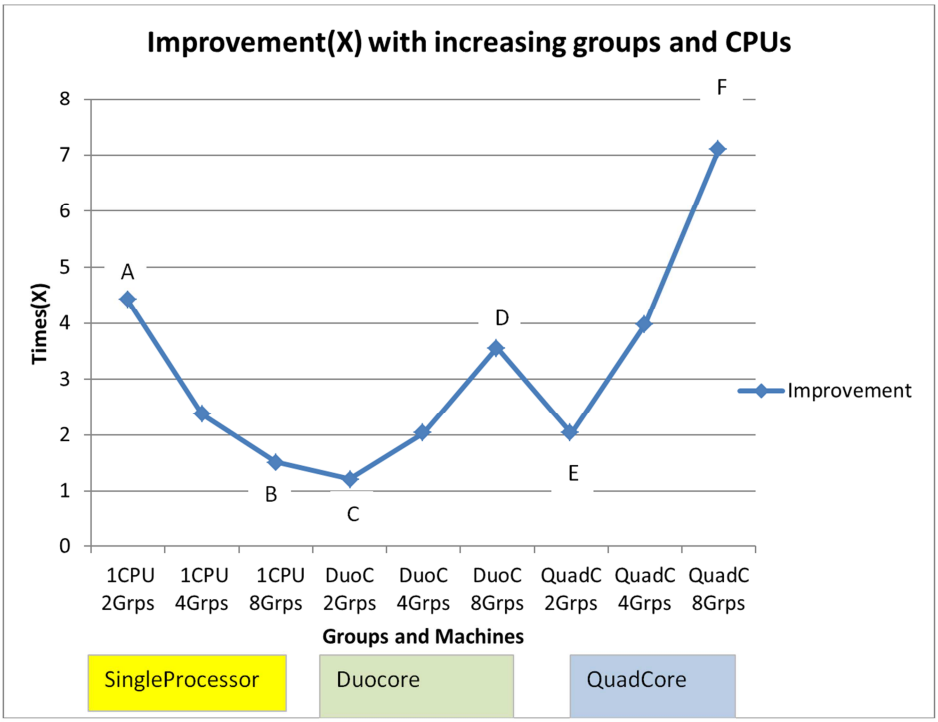


***Figure 3.*** *Performance trend among between the systems.*

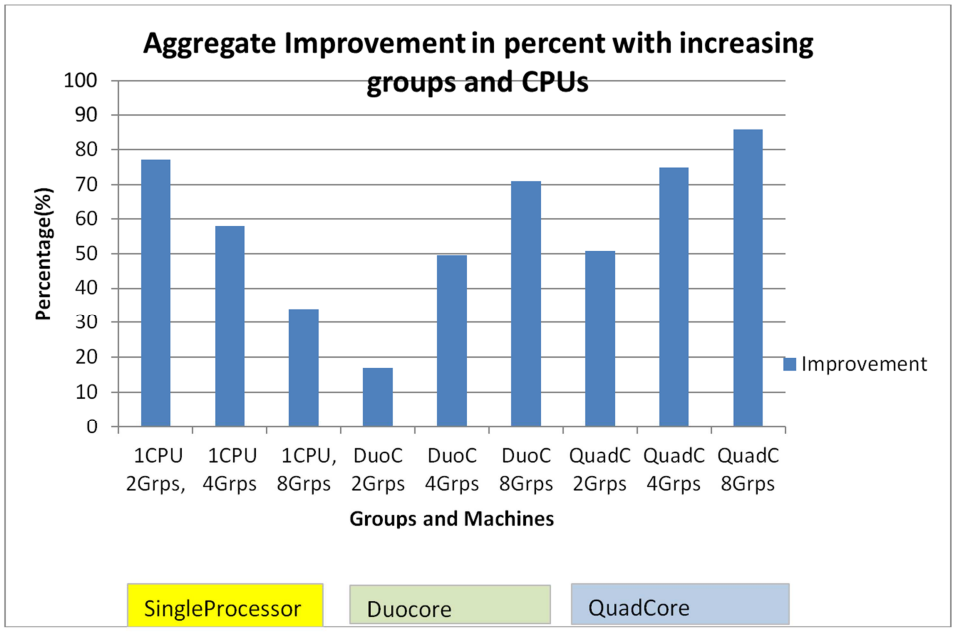


***Figure 4.*** *Aggregate performance.*

***Table 3.*** *Experiment results from three computing platforms.*

| No of Jobs | Quadcore | Single Processor | | | Duocore | | | Quadcore | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | MinMin | 2Grps | 4Grps | 8Grps | 2Grps | 4Grps | 8Grps | 2Grps | 4Grps | 8Grps |
| 1000 | 403 | 2360 | 1094 | 594 | 375 | 319 | 95 | 189 | 111 | 78 |
| 2000 | 1775 | 7907 | 3876 | 2031 | 1360 | 617 | 331 | 759 | 337 | 196 |
| 3000 | 4203 | 17031 | 9063 | 4360 | 2628 | 1334 | 609 | 1543 | 648 | 396 |
| 4000 | 7431 | 28468 | 13656 | 9359 | 4665 | 2138 | 1575 | 2570 | 1297 | 671 |
| 5000 | 9903 | 40453 | 22672 | 12985 | 7680 | 3832 | 2323 | 4339 | 1928 | 1178 |
| 6000 | 12984 | 59781 | 32578 | 19062 | 9607 | 5481 | 2927 | 6433 | 3089 | 1568 |
| 7000 | 16872 | 79499 | 42047 | 26234 | 14322 | 7888 | 4864 | 9079 | 4315 | 2361 |
| 8000 | 21956 | 94781 | 54172 | 34453 | 17156 | 9967 | 5672 | 11261 | 5758 | 2855 |
| 9000 | 27778 | 119609 | 67438 | 41250 | 22976 | 12796 | 7273 | 14340 | 7082 | 4506 |
| 10000 | 34570 | 157970 | 82423 | 55282 | 27406 | 15921 | 9080 | 17385 | 10593 | 6379 |
| Total | 137875 | 607859 | 329019 | 205610 | 108175 | 60293 | 34749 | 67898 | 35158 | 20188 |
| Avg | 13787.5 | 60785.9 | 32901.9 | 20561 | 10817.5 | 6029.3 | 3474.9 | 6789.8 | 3515.8 | 2018.8 |

***Table 4.*** *Combined Improvement Analysis in Multiples (X).*

| Combined Improvement Analysis in Multiples | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Single Processor | | | Duocore | | | Quadcore | | |
| | 2Grps | 4Grps | 8Grps | 2Grps | 4Grps | 8Grps | 2Grps | 4Grps | 8Grps |
| 1 Threads | 4.37 | 2.36 | 1.53 | 1.29 | 2.22 | 3.95 | 2.03 | 4.07 | 6.96 |
| 2 Threads | 4.41 | 2.39 | 1.49 | 1.27 | 2.29 | 3.97 | 2.03 | 3.92 | 6.83 |
| 4 Threads | 4.43 | 2.38 | 1.52 | 1.07 | 1.61 | 2.76 | 2.03 | 3.96 | 7.52 |
| Aggreg. Improvement | 4.40 | 2.37 | 1.51 | 1.21 | 2.04 | 3.56 | 2.03 | 3.98 | 7.10 |

***Table 5.*** *Combined Improvement Analysis in percentage (%).*

| Combined Improvement Analysis in Percentage | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Single Processor | | | Duocore | | | Quadcore | | |
| | 2Grps | 4Grps | 8Grps | 2Grps | 4Grps | 8Grps | 2Grps | 4Grps | 8Grps |
| 1 Threads | 77.13 | 57.56 | 34.80 | 22.26 | 54.87 | 74.69 | 50.82 | 75.42 | 85.64 |
| 2 Threads | 77.32 | 58.10 | 32.94 | 21.54 | 56.27 | 74.80 | 50.75 | 74.50 | 85.36 |
| 4 Threads | 77.44 | 57.93 | 34.10 | 6.87 | 37.76 | 63.78 | 50.83 | 74.73 | 86.70 |
| Aggregate Improvement | 77.30 | 57.86 | 33.95 | 16.89 | 49.64 | 71.09 | 50.80 | 74.88 | 85.90 |

### *3.7. General Discussion on the Results*

The experiments were conducted on a different set of computer systems running on CPUs from a different family with different internal architecture, different clock speeds, different memory access patterns, different cache coherence, different levels of hit rate. As a result, the overall result cannot be normalized due to differences in factors that determine the performance of a computer's processor. However, the results and measured performance improvement were strongly correlated (0.99), also, the R-Squared value of the performance improvement on each machine was high (0.9x), thus indicating a high degree of correctness and reliability.

From the analysis, we demonstrated that by placing machines and independent jobs into groups before simultaneously multischeduling in parallel enhances the overall scheduling time. We find that with increased groups, the performance of the grouping method continues to improve against the ordinary MinMin on the multicore systems. We also find that there was a general increase in performance as the number of cores increases.

Relying on the underlying multicore system for parallelism is not the best for scheduling and processing - especially when the algorithm is sequential. However, our findings showed that the ordinary MinMin algorithm is scalable and performed better on the quadcore machine, then on the duocore machine and least on the single processor machine. Also, the MinMin on the quadcore performed better than the group method executed on a single processor system. This is because the MinMin algorithm scaled up on the quadcore machine while the single processor system offered but minimal parallelism for threads execution required by the group method.

On the multicore machines, the group method performed far better than the ordinary MinMin correspondingly and also when measured against the best MinMin result obtained from the quadcore. Although all the machines are not timed at the same speed, we can conclude from the analysis that the group method increases parallelism which resulted in improved performance as the number of groups increases and also as the number of cores increases.

The results and improvement analysis were highly correlated (0.99) and a plot of performance improvement on corresponding machines yielded a linear trend with increasing function and also with a high R-Squared value of 0.99. This high R-Squared value indicates a very good fit. It also indicates the reliability and predictability of the method. Generally, it shows that using more groups will continue to improve the performance of scheduling algorithms on multicore systems and also that machines with more cores will guarantee better performance over machines with fewer cores.

# 4. Recommendations

Results and analysis have shown that serial algorithms do not fully exploit parallelism on multicore systems. It has also been shown that serial algorithms are not completely parallelizable, for the grid to attain its goals, efforts aimed at scaling the scheduling and processing of Grid jobs in line with the ready benefits of the multicore systems should be encouraged.

The result and analysis from this experiment indicate that grouping jobs and machines before scheduling greatly enhances improvement on multicore systems. Therefore, we recommend that performance improvement methods like the group method be integrated into job scheduling and processing in the Grid.

# 5. Conclusion

This work combined the experiments in our previous work using the group method on three different computing platforms. The aim is to enhance the scheduling of Grid jobs by exploiting parallelism on multicore systems. On the single processor, the group method recorded 50%, 73% and 83%. Or 2.03, 3.76, and 6.01 Times improvement over the MinMin when executed with 2, 4 and 8 groups respectively.

On the duocore machine, the group method performed 50%, 72% and 84% or 2.04, 3.66, and 6.35 Times better than the MinMin when executed with 2, 4 and 8 groups respectively.

On the quadcore, the group method performed 50%, 74% and 85% or 2.03, 3.92, and 6.88 Times better than the MinMin algorithm.

All results are strongly correlated and the trendline through the performance graph has a very high R-squared value – representing a perfect fit.

We conclude that the group method enhances the performance of multicore systems. We also conclude that the group method directly improves performance of multicore systems as the number of cores increases.

# Future Thoughts

This research enhances the scheduling of Grid jobs using a novel independent machine and job grouping method and multischeduling. Load balancing was ensured between groups using the proportion of total processing power of machines in the group.

The experiment was executed with a sequential (MinMin) scheduling algorithm. The algorithm was found to be scalable on the multicore systems but scheduling was greatly improved using the group method. In the future, the work can be extended to parallel scheduling algorithms.

This experiment targets the Grid and also uses data from the grid workload archive, giving the close relationship between grid computing and cloud computing, in the future, we hope to adapt the work for the cloud.

The result cannot be standardized because the experiment was executed on systems with different features and family of processors, to standardize the result, the experiment should be executed on a set of systems from the same family of CPU that shares the same features.

Based on the linearity of performance improvement with increasing groups through the computing platforms, we intend to explore further the number of groups used and extrapolate the limit to be able to proffer other solutions.

From the result and analysis, we intend to carry out a performance measure of the three systems by keeping some parameters constant while varying the machine. This will give an idea of the performance measure between the three systems.

# References

[1] A. Chervenak, I. Foster, C. Kesselman, C. and Salisbury, and S. Tueke, "The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets," *Journal of Network and Computer Applications*, vol. 23, no. 3, pp. 187–200, 2000, Accessed: Mar. 04, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804500901103.

[2] D. Klusáček, H. Rudová, R. Baraglia, M. Pasquali, and G. Capannini, "Comparison Of Multi-Criteria Scheduling Techniques," in *Grid Computing*, Springer US, 2008, pp. 173–184.

[3] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of scheduling techniques for addressing shared in multicore processors," *ACM Reference Format*, vol. 45, no. 4, Nov. 2012, doi: 10.1145/2379776.2379780.

[4] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *AFIPS Conference Proceedings - 1967 Spring Joint Computer Conference, AFIPS 1967*, Apr. 1967, pp. 483–485, doi: 10.1145/1465482.1465560.

[5] C. Kessler, U. Dastgeer, and L. Li, "Optimized Composition: Generating Efficient Code for Heterogeneous Systems from Multi-Variant Components, Skeletons and Containers," *Skeletons and Containers. arXiv preprint arXiv*, vol. 1405.2915, May 2014, Accessed: Mar. 06, 2020. [Online]. Available: http://arxiv.org/abs/1405.2915.

[6] J. Larus, "Spending Moore's dividend," in *Communications of the ACM*, May 2009, vol. 52, no. 5, pp. 62–69, doi: 10.1145/1506409.1506425.

[7] M. Gebremedhin, "Automatic and Explicit Parallelization Approaches for Equation Based Mathematical Modeling and Simulation," 2018, Accessed: Apr. 21, 2021. [Online]. Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2:1265975.

[8] H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, and B. Chapman, "High performance computing using MPI and OpenMP on multi-core parallel systems," *Parallel Computing*, vol. 37, no. 9, pp. 562–575, 2011, doi: 10.1016/j.parco.2011.02.002.

[9] B. Mustafa, S. and Rafiya, and A. Waseem, "Parallel Implementattion of Doolittle Algorithm using OpenMP for multicore machines," in *2015 IEEE International Advance Computing Conference*, 2015, pp. 575–578.

[10] P. Tendulkar, "Mapping and Scheduling on Multi-core Processors using SMT Solvers," 2014.

[11] G. T. Abraham, A. and James, and N. Yaacob, "Priority-grouping method for parallel multi-scheduling in Grid," *Journal of Computer and System Sciences*, vol. 81, no. 6, pp. 943–957, 2015, doi: 10.1016/j.jcss.2014.12.009.

[12] S. A. Mirsoleimani, A. Karami, and F. Khunjush, "A parallel memetic algorithm on GPU to solve the task scheduling problem in heterogeneous environments," in *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference*, 2013, pp. 1181–1188, doi: 10.1145/2463372.2463518.

[13] F. Peper, "The End of Moore's Law: Opportunities for Natural Computing?," *New Generation Computing*, vol. 35, no. 3, pp. 253–269, Jul. 2017, doi: 10.1007/s00354-017-0020-4.

[14] B. Schauer, "Discovery Guides Multicore Processors-A Necessity," 2008. Accessed: Mar. 05, 2020. [Online]. Available: http://www.netrino.com/node/91.

[15] G. Bell, "Bell's law for the birth and death of computer classes: A theory of the computer's evolution," *IEEE Solid-State Circuits Society Newsletter*, vol. 13, no. 4, pp. 8–19, 2008.

[16] S. Eyerman and L. Eeckhout, "Modeling critical sections in Amdahl's law and its implications for multicore design," in *Proceedings - International Symposium on Computer Architecture*, 2010, pp. 362–370, doi: 10.1145/1815961.1816011.

[17] N. Shavit, "Data structures in the multicore age," *Communications of the ACM*, vol. 54, no. 3, pp. 76–84, Mar. 2011, doi: 10.1145/1897852.1897873.

[18] L. Silva, "Computing data cubes over GPU clusters.," 2018, Accessed: Mar. 04, 2020. [Online]. Available: https://www.monografias.ufop.br/handle/35400000/1527.

[19] Y. Ngoko and D. Trystram, "Enhancing the undergraduate curriculum, Performance, concurrency and programming on modern platform," in *Topics in Parallel and Distributed Computing*, vol. 1st Edition, 2018, pp. 337-undefined.

[20] R. E. N. Bin, S. Balakrishna, Y. Jo, S. Krishnamoorthy, K. Agrawal, and M. Kulkarni, "Extracting SIMD parallelism from recursive task-parallel programs," *ACM Transactions on Parallel Computing*, vol. 6, no. 4, pp. 1–37, Dec. 2019, doi: 10.1145/3365663.

[21] S. K. Roy, R. Devaraj, A. Sarkar, K. Maji, and S. Sinha, "Contention-aware optimal scheduling of real-time precedence-constrained task graphs on heterogeneous distributed systems," *Journal of Systems Architecture*, vol. 105, p. 101706, May 2020, doi: 10.1016/j.sysarc.2019.101706.

[22] M. M. Javanmard, Z. Ahmad, M. Kong, L.-N. Pouchet, R. Chowdhury, and R. Harrison, "Deriving parametric multi-way recursive divide-and-conquer dynamic programming algorithms using polyhedral compilers," in *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization*, Feb. 2020, pp. 317–329, doi: 10.1145/3368826.3377916.

[23] Q. Tang, L.-H. Zhu, J. Lian, L. Zhou, and J.-B. Wei, "An efficient multi-functional duplication-based scheduling framework for multiprocessor systems," *The Journal of Supercomputing*, pp. 1–26, Feb. 2020, doi: 10.1007/s11227-020-03208-y.

[24] B. N., Chandrashekhar, H. A., and Sanjay, and T. Srinivas, "Performance Analysis of Parallel Programming Paradigms on CPU-GPU Clusters," in *International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, 2021, pp. 646–651, Accessed: Apr. 19, 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9395977/.

[25] Y. Zou, Y. Zhu, Y. Li, F.-X. Wu, and J. Wang, "Parallel computing for genome sequence processing," *Briefings in Bioinformatics*, Apr. 2021, doi: 10.1093/bib/bbab070.

[26] D. Tran, S. Aslam, N. Gorius, and G. Nehmetallah, "Parallel Computation of CRC-Code on an FPGA Platform for High Data Throughput," *Electronics*, vol. 10, no. 7, p. 866, 2021, Accessed: Apr. 23, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/7/866.

[27] M. Salvana *et al.*, "High Performance Multivariate Geospatial Statistics on Manycore Systems," *IEEE Transactions on Parallel and Distributed Systems*, 2021, doi: 10.1109/TPDS.2021.3071423.

[28] F. Baig, "High Performance Spatial and Spatio-Temporal Big Data Processing," 2021. Accessed: Apr. 23, 2021. [Online].

[29] M. J. Bridges, N. Vachharajani, Y. Zhang, T. Jablin, and D. I. August, "Revisiting the sequential programming model for the multicore era," *IEEE Micro*, vol. 28, no. 1, pp. 12–20, Jan. 2008, doi: 10.1109/MM.2008.13.

[30] S. H. Fuller and L. I. Millett, *The future of computing performance: Game over or next level?* National Academies Press, 2011.

[31] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit, "On Advantages of Grid Computing for Parallel Job Scheduling," in *2nd IEEE/ACM International symposium on Cluster Computing and the Grid*, 2002, pp. 339–39, Accessed: Mar. 09, 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1540439/.

[32] N. Muthuvelu, J. Liu, L. Soe, S. Venugopal, A. Sulistio, and R. Buyya, "A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids," in *Australian Workshop on Grid computing and e-research*, 2005, pp. 41–48, Accessed: Mar. 09, 2020. [Online]. Available: https://dl.acm.org/citation.cfm?id=1082297.

[33] V. K. Soni, R. and Sharma, and K. Mishra, Manoj, *Grouping-based job scheduling model in grid computing*, vol. 41. 2010.

[34] G. T. Abraham, A. and James, and N. Yaacob, "Group-based Parallel Multi-scheduler for Grid computing," *Future Generation Computer Systems*, vol. 50, pp. 140–153, 2015, doi: 10.1016/j.future.2015.01.012.

[35] G. T. Abraham, "Group-based parallel multi-scheduling methods for grid computing," Coventry University, 2016.

[36] G. T. Abraham and E. F. Osaisai, "Parallel Scheduling of Grid Jobs on Duo-core Systems Using Grouping Method," *International Journal of Current Research*, vol. in Publica.

[37] G. T. Abraham, E. F. and Osaisai, and N. Dienagha "Parallel Scheduling of Grid Jobs on Quadcore Systems Using Grouping Methods," *Asian Journal of Research in Computer Science*, vol. 8, no. 4, pp. 21–34, 2021, [Online]. Available: https://doi.org/10.9734/ajrcos/2021/v8i430207.

[38] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM*, vol. 24, no. 2, pp. 280–289, 1977, Accessed: May 06, 2021. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/322003.322011.

[39] S. Nesmachnow and M. Canabé, "GPU implementations of scheduling heuristics for heterogeneous computing environments," 2011, Accessed: Mar. 09, 2020. [Online]. Available: http://sedici.unlp.edu.ar/handle/10915/18652.

[40] M. Canabe and S. Nesmachnow, "Parallel implementations of the MinMin heterogeneous computing scheduler in GPU," *CLEI Electronic Journal*, vol. 15, no. 3, pp. 8–8, 2012, Accessed: Feb. 10, 2020. [Online]. Available: http://www.scielo.edu.uy/scielo.php?pid=S0717-50002012000300009&script=sci_arttext&tlng=pt.

[41] K. Etminani and M. Naghibzadeh, "A min-min max-min selective algorihtm for grid task scheduling," in *in 2007 3rd IEEE/IFIP International Conference in Central Asia on Internet*, 2007, pp. 1–7, Accessed: May 06, 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4401694/.

[42] Freund Richard, Taylor Kidd, Hensgen Debbie, and Lantz Moore, "SmartNet: a scheduling framework for heterogeneous computing," in *Second International Symposium on Parallel Architectures, Algorithms and Networks (IEEE 96)*, 1996, pp. 514–521, Accessed: May 06, 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/509034/.

[43] M. Lavanya, B. and Shanthi, and S. Saravanan, "Multi objective task scheduling algorithm based on SLA and processing time suitable for cloud environment," *Computer Communications*, vol. 151, pp. 183–195, 2020, Accessed: May 06, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S014036641930492X.

[44] M. Maheswaran, S. Ali, H. Siegel, D. and Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107–131, 1999, Accessed: May 06, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731599915812.

[45] S. K. Mishra and B. Sahoo, "Load balancing in cloud computing: A big picture," *Journal of King Saud University - Computer and Information Sciences*, vol. 32, no. 2, pp. 149–158, 2020, Accessed: May 06, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157817303361.

[46] F. Pinel, B. Dorronsoro, and P. Bouvry, "Solving very large instances of the scheduling of independent tasks problem on the GPU," *Journal of Parallel and Distributed Computing*, vol. 73, no. 1, pp. 101–110, 2012, doi: 10.1016/j.jpdc.2012.02.018.

[47] Zhou, Z., Li, F., Zhu, H., Xie, H., Jemal, H. A. and, & Morshed, U. C. (2020). An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments. Neural Computing and Applications, 32 (6), 1531–1541. https://link.springer.com/article/10.1007/s00521-019-04119-7

[48] Zhuravlev, S., Saez, J. C., Blagodurov, S., Fedorova, A., & Prieto, M. (2012). Survey of scheduling techniques for addressing shared in multicore processors. ACM Reference Format, 45 (4). https://doi.org/10.1145/2379776.2379780