Coursework Declaration and Feedback Form

The Student should complete and sign this part

Student Number: 2727018y	Student Name: Wenbo Yang
Programme of Study (e.g. MSc in Electronics and Electrical Engineering): MSc in Computer System Engineering	
Course Code: ENG5059P	Course Name: MSc Project
Name of **First** Supervisor: Paul Harvey	Name of **Second** Supervisor: GTA11
Title of Project: Exploring Modularity in MapReduce Distributed Computing Framework	

Declaration of Originality and Submission Information

I affirm that this submission is all my own work in accordance with the University of Glasgow Regulations and the School of Engineering requirements Signed (Student) : wenbo yang	E N G 5 0 5 9 P
Date of Submission : August 14th, 2023	

Feedback from Lecturer to Student – to be completed by Lecturer or Demonstrator	
Grade Awarded: Feedback (as appropriate to the coursework which was assessed):	
Lecturer/Demonstrator:	Date returned to the Teaching Office:

Exploring Modularity in MapReduce Distributed Computing Framework

University of Glasgow

Name: Wenbo Yang
GUID: 2727018y

Abstract

As technology advances, software engineers confront intricate problems, necessitating efficient and scalable automation solutions. The emergence of big data and cloud computing has underscored the importance of adeptly processing and allocating vast amounts of data. MapReduce, initiated by Google, has been pivotal in this context, offering a robust framework for data-intensive distributed computing. However, developers face challenges within the MapReduce ecosystem, especially regarding the repetitive design and management of task distribution strategies. This paper presents a pioneering approach by introducing adaptive modularization within the MapReduce framework, aiming to simplify programming and augment the effectiveness of distributed processing systems. Through a simulated environment mimicking MapReduce functionality and encompassing four different allocation algorithms, the potential of this modular strategy is demonstrated. Evaluative metrics like standard deviation in task counts per node, processing units used, and words processed per node reveal significant insights. Results show different algorithm modules perform differently in different situations and also emphasize the paramount importance of choosing optimal algorithms tailored to specific system requirements. This research represents a leap towards smarter, more efficient, and flexible distributed computing systems.

Acknowledgments

I express my profound gratitude to all the people who help me. It was their presence that allowed me to successfully complete the report.

I must first and foremost express my gratitude to my mentor. He has not only given me important academic advice throughout the entire study process, but he has also consistently encouraged and supported me. Every time I faced challenges or felt lost, he patiently clarified my doubts and steered me back on the right path. His rigorous expectations combined with unwavering encouragement have led to my continuous academic advancements and instilled in me a strong confidence and anticipation for future research endeavors.

Secondly, I wish to thank my college. I'm grateful for being given such a conducive academic platform and a superior research environment, enabling me to delve deep into my studies and exploration. The diverse resources and facilities provided by the institution have been instrumental in the smooth progression of my work.

In conclusion, my heartfelt appreciation goes out to my friends and colleagues for their invaluable assistance and backing. Their aid was instrumental, and without it, finalizing this report might have been an insurmountable task.

Once again, my sincere appreciation to all.

Table of Contents

1. Introduction

In today's computer field, with the continuous development of technology, software engineers are facing increasingly complex problems. An efficient and scalable automation solution is particularly important.

The rise of big data and cloud computing has reshaped how we manage vast data volumes. Efficiently processing and allocating tasks on a grand scale is now pivotal to this shift. Among various models tailored for processing data on such a scale, the MapReduce framework has become a favored option because of its innate capacity for distributed and concurrent operations [1].

MapReduce was launched by Google and is a widely used framework in the realm of data-intensive distributed computing, particularly in relation to batch jobs [2]. It provides a simplified programming model that enables scalable computing on large datasets to be easily parallelized [1].

Nonetheless, there remains a significant challenge for developers working within the MapReduce framework, which stems from the need to repetitively design and manage task distribution strategies for different applications. Consequently, it has become clear that a more modular, adaptive strategy would help to simplify programming and boost distributed processing systems' overall effectiveness.

Modularization, as applied in the field of computer science, has the capacity to enhance adaptability and diminish intricacy within software applications. Software developers could focus on creating components that meet specific needs without the need to understand the entire system by dividing the system into smaller and interchangeable modules. The efficiency of data processing tasks. The combination of modularization and MapReduce promises unprecedented levels of flexibility and scalability.

In this paper, we delve into the potential of adaptive modularization within the MapReduce framework. We introduce a simulated environment that mimics the functionality of MapReduce, comprising a Node Manager for controlling nodes and a Task Scheduler for assigning tasks.

Furthermore, we investigate the performance implications of our approach by analyzing metrics including the standard deviation in task counts per node, processing units used, and words processed per node. These steps provide understanding about the system's flexibility and avenues for additional enhancement.

The objective of this paper is to present an adaptive, modular strategy for task distribution within the MapReduce framework and to highlight its capability to address programming obstacles, elevate flexibility, and boost overall system efficiency. This research signifies an advancement in the pursuit of more intelligent, efficient, and flexible distributed computing systems.

2. Literature review

2.1 MapReduce

Data processing landscapes have changed with the introduction of Google's MapReduce framework in 2004. A large data set can be processed and generated using MapReduce, a programming model, and its associated implementation [3]. The model was designed to simplify the complexities associated with processing vast amounts of data across distributed systems. Google MapReduce processes over 10 petabytes of information per day, according to public statistics [3]. MapReduce, by simplifying parallelization, data distribution, and fault resilience, offers a high-level programming model, enabling developers to focus primarily on the core logic of their software.

MapReduce has two primary functions: **Map** and **Reduce**. Accepts a pair of input values and outputs a series of intermediate key/value pairs that were written by the user These intermediate pairs are then processed by the Reduce function, which merges all intermediate values associated with the same intermediate key [4].

The impact of MapReduce extends far beyond its initial application at Google. It has become the foundation for many big data processing systems. Hadoop, a recently built open-source version, is becoming more widely used in both industry and academia [5]. And Hadoop is used by more than 70 companies [6]. MapReduce's extensive adoption has played a pivotal role in the evolution and growth of the big data sector, enabling the efficient processing of enormous datasets on standard hardware clusters.

The performance, effectiveness, and adaptability of MapReduce frameworks have been improved over time with the introduction of several optimizations and upgrades. Innovations like data locality, speculative execution, and combiners have enhanced the model's performance, while the introduction of new abstractions like Apache Spark's RDD and Data Frame have improved its flexibility.

While the MapReduce paradigm has brought about noteworthy advancements, it still evolves to cater to the escalating needs of data-centric applications, making it a vibrant arena for research and progress in distributed computing. The exploration on this basis should not stop either.

2.2 Modularity

Modularity, a fundamental concept in many disciplines. Modularization is a technique for handling complexity that makes it easier to link and manage systems by dividing them into distinct sections and connecting them via common interfaces [7]. In computer science, it pertains to the design approach where a system is decomposed into smaller, autonomous units or modules, with each serving a distinct purpose. The aim of modularity is to simplify complexity, facilitate system comprehension, and boost maintainability and scalability.

Software engineers may independently work on specific modules thanks to modular architecture, which improves parallel development and cuts down on testing and debugging time. Modules encapsulate functionality and expose well-defined interfaces for communication, hiding their internal complexity [8]. This practice of information hiding contributes to robustness, as changes in one module shouldn't affect others, given the interface remains consistent.

Furthermore, modularity fosters code reuse, a central tenet in software development. A well-designed module may be reused across several applications, cutting down on development time and effort. This reuse also enhances reliability, as modules tested and used in diverse contexts are more likely to be dependable.

Modularity also has profound implications in emerging fields like microservices in software architecture and function-as-a-service in cloud computing. These paradigms, built on modularity, allow for better scaling, isolation, and independent deployment, emphasizing the enduring relevance of modularity in evolving technological landscapes.

2.3 Modularity in MapReduce

The MapReduce framework already embodies modularity. The input data is divided into independent chunks that are processed by the map tasks, acting as standalone modules. The framework then consolidates the map results and routes them to reduce tasks, which again, operate as independent modules. This design not only simplifies parallel processing but also enhances scalability, as additional map and reduce tasks can be easily introduced based on data volume and computational complexity [9].

From a development standpoint, each MapReduce job can be treated as a module. These jobs can be chained together to create complex data processing pipelines, promoting code reuse and maintainability. The encapsulation of each job enhances code robustness, as changes in one job should not affect others, given that the data interfaces remain consistent.

Moreover, modularity surfaces in the execution environment of MapReduce as well. The task tracker and job tracker components in a Hadoop-based MapReduce system, for example, function independently but in unison, maintaining the system's overall health and facilitating task scheduling.

Resilient Distributed Datasets (RDDs) and Data Frames are extra levels of modularity offered by sophisticated MapReduce-inspired frameworks like Apache Spark, giving programmers more adaptable and effective abstractions for handling enormous amounts of data [10].

2.4 Bin packing problem

The Bin Packing Problem (BPP) is a classic optimization problem that has been extensively studied in the field of operations research and theoretical computer science [11]. Given a set of items with varying sizes and a fixed-size bin, the objective is to pack these items into the fewest bins possible [12]. The BPP has various real-world applications, ranging from cargo loading to computer memory allocation.

The early research on BPP focused on exact algorithms. Martello and Toth (1990) are two notable names that proposed branch and bound algorithms for the one-dimensional bin packing [13]. While these exact methods provide optimal solutions, they tend to be computationally infeasible for large instances.

Due to the NP-hard nature of the problem, heuristics and approximation algorithms have been more commonly adopted for real-world applications. Several heuristics have been proposed, including:

First Fit (FF): Places each item in the first bin in which it fits.
Best Fit (BF): Places each item in the tightest bin where it can be accommodated.
Next Fit (NF): Uses the current open bin until it is full and then opens a new one.
Worst Fit (WF): Places each item in the bin with the most remaining space, assuming it can be accommodated.

The basic BPP has seen numerous extensions and variants. The multiple bin size problem, where bins can have different capacities, and the two-dimensional bin packing, where items are rectangular and bins are two-dimensional, are such examples. There's also dynamic bin packing, where items can arrive and depart over time [14].

The Bin Packing Problem remains an essential topic in combinatorial optimization. Its theoretical significance, combined with its numerous practical applications, ensures its prominence in research. As computational capabilities advance, it's expected that new methods and applications related to BPP will continue to emerge.

2.5 Challenges

Despite the remarkable progress made through the fusion of MapReduce and modularity in big data processing, several challenges remain to be addressed:

- Balancing Algorithm Optimization with Modularity: With the introduction of new algorithms, like those applied to the Bin Packing Problem, a key challenge lies in integrating and optimizing these algorithms while maintaining effective modularity.

- Complexity of Resource Allocation: In large distributed systems, specific resource allocation strategies (e.g., FF, BF, NF, WF) might underperform due to the dynamic and intricate nature of the system. Modularizing effectively under these conditions poses a significant challenge.

- Adaptability of Modularity: While modularity aids in simplifying development and maintenance, it might complicate certain specific optimizations when they span across modules. Maintaining efficient inter-module communication and avoiding performance bottlenecks are ongoing challenges.

- Dynamic Adjustments and Scalability: The ability to dynamically adjust and maintain efficient performance in a MapReduce system when faced with varying data flows and computational demands, all without sacrificing the benefits of modularity, stands as a core challenge.

To address these challenges, this paper will adopt an experimental approach. By constructing a simulated MapReduce framework, we aim to create an environment conducive to testing and understanding the implications of modularity in resource allocation. Within this environment, modularized allocation algorithms will be implemented and evaluated. This experimental setup not only facilitates a deeper understanding of the interplay between MapReduce and modularity but also allows for tangible solutions to the challenges identified. Through this hands-on approach, we aspire to bridge the gap between theoretical concepts and their practical applications, providing valuable insights into the optimization and adaptability of modularized allocation strategies in distributed systems.

3. Environment and Approach

3.1 Environment setup

Before modular design can be implemented, an environment that simulates the MapReduce distributed framework must first be constructed. This environment emulates a MapReduce distributed framework, allowing for the testing of different allocation algorithms under a unified setting. The described framework primarily comprises three classes: the Node class, Task Scheduler class, and Node Manager class. The process of task allocation, execution of MapReduce functions on data, and data storage is also elucidated. Here, to simplify the experiment, we have used the mapreduce function to replace the parts of mapping, shuffling, and reducing. It is important to note that since the experiment is a test of the task distribution module, this modification will not affect the accuracy of the experiment.

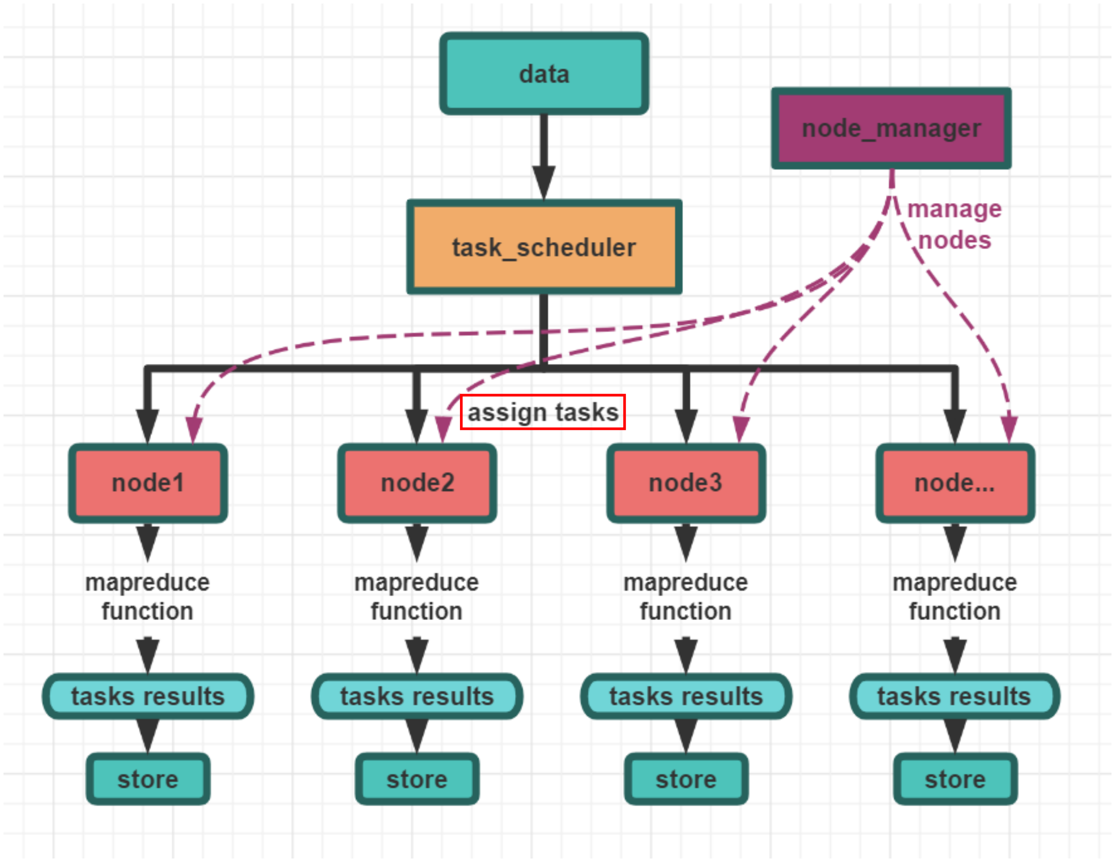


Figure 1: MapReduce Simulation Distributed Framework Flowchart

The framework depicted in Figure 1 was constructed using Python code. The experiment primarily focuses on the modular study of the highlighted sections. Four distinct allocation algorithm modules were designed. Upon the arrival of tasks to the Task Scheduler, they are allocated to nodes for execution influenced by these allocation algorithms. The experiment revolves around the performance differences of these four allocation algorithm modules within this framework.

The tasks that need to be processed are initially transferred to the Task Scheduler. The Task Scheduler has the capability to invoke various allocation algorithms to delegate these tasks to different nodes. This assignment of tasks to nodes is the first crucial step in the process. After the tasks assigned, nodes execute the MapReduce function to process the data. This stage is referred to as mapping. Following the data processing, the data is stored in the local memory of each node. The functionality of this stage goes beyond mere data storage. It ensures that the processed data is readily accessible for further operations, facilitating efficient data management. The Node Manager class is a vital component of this framework. Its primary role involves the management of node creation, initiation, and termination. This class maintains the smooth operation of the system by managing its fundamental units, i.e., the nodes.

The simulation environment facilitates the testing of different allocation algorithms within a MapReduce distributed framework. By categorizing functions into the Node, Task Scheduler, and Node Manager classes, the framework allows for efficient allocation and processing of tasks, as well as effective node management. This approach, therefore, forms a solid foundation for further modularized design, opening avenues for improved efficiency and scalability in distributed computing systems.

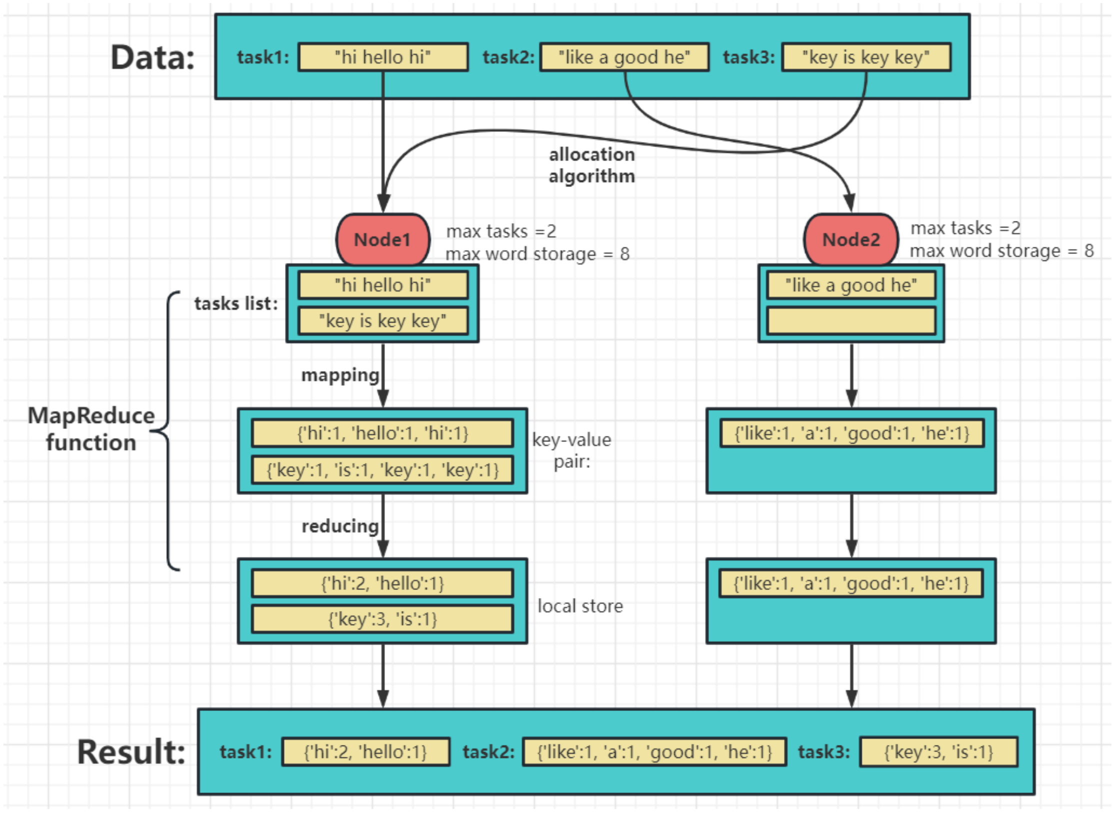


Figure 2: MapReduce function Flowchart

Figure 2 vividly illustrates the process of data handling. After assigning tasks to nodes, the tasks undergo a mapping process where the words within the tasks are converted into key-value pairs. These key-value pairs are then consolidated via a reducing operation to produce the results.

3.2 Code function

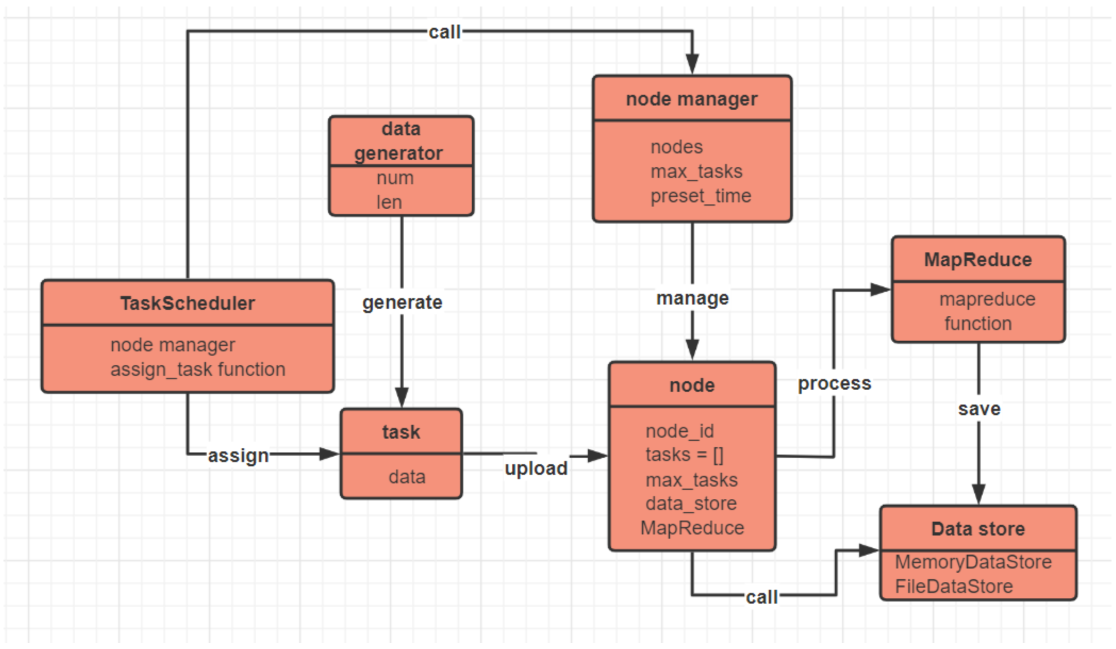


Figure 3: Code Class Structure Diagram

Figure 3 displays the structure of the various classes and the relationships between them. In the following, a detailed code-level analysis of each class will be provided.

- MapReduce: mapreduce function is used to simulate a function that runs in the MapReduce framework. This process is primarily divided into two steps: mapping and reducing. In the first step, mapping, the function is designed to accept a discrete unit of data (in the form of a string). It then disassembles this string into its individual word components. For each word parsed from the string, the function crafts a tuple. in each of these tuples, the first component is the word, and the second component is represented count one. This value means one occurrence of the corresponding word. After this stage, this function returns a list of all tuples. The subsequent step, reducing, operates differently. This function uses a key (a word) and a list of values (these values are the list of 1's corresponding to the word). The function calculates the sum of values and stores in a list, and then returns a tuple. The first component of this tuple is the word, and the second is the words frequency.

  Although, the purpose of this MapReduce program is to analyze the dataset (in this case, a string of text). It aims to calculate and present the frequency of each distinct word across the entirety of the data. This is a typical and commonly observed use case in the context of MapReduce operations and is more formally known as word count statistics or word frequency analysis. This function exists only to restore the complete structure in the MapReduce framework, and its own functionality is not overly related to this project.

- Node class: The node class inherits from the threading. Thread class, which indicates that each Node instance is a separate thread. The class can take 5 parameters which are node ID, maximum number of tasks, maximum time, number of CPU cores, and storage object. The run method of this class is the subject function of the thread. In each loop, it first tries to get and remove a task from the task list. Then, if there is a task, it executes that task (by creating a MapReduce object and executing the task as input) and saves the result to the data storage object. Overall, each node runs in its own thread and can independently get tasks from the task list and execute them. The Node object here represents some node that processes the tasks and is used to simulate processing nodes in distributed computing.

- NodeManager class: This class manages the created nodes through different methods. start_node method starts all the Node objects. close_node method first checks if there are still tasks to be completed on all the nodes. If there is, then it pauses (sleeps) the program for a second and then checks again until all tasks have been completed for all nodes. It then tells all Node objects that they should stop (by setting their should_stop property to True). Finally, it waits for all Node objects to finish shutting down, which is done by calling the join method of the Node object. get_nodes method returns a list of all Node objects managed by the NodeManager.

- TaskScheduler class: This class is used to assign tasks to a collection of Node objects managed by a NodeManager. Within the framework of this class, various task distribution methodologies are incorporated, comprising strategies labeled as "best fit", "worst fit", "first fit", and "next fit". In this context, these modules are deployed for the allocation of tasks.

1. The assign_task_ff method follows the First Fit allocation principle which places each item into the first bin it can fit into and opens a new bin if no current bin can accommodate the item [15]. In this experiment its algorithmic logic is to traverse from the first node and store the task in the node when the space required by the task is less than the remaining space of the node. The algorithm logic is to judge whether the processing unit and the number of words is smaller than the node's remaining space for throughput and the number of remaining words. This strategy can quickly allocate nodes with high utilization.

2. The assign_task_bf method follows the best fit allocation principle, and in this experiment its algorithmic logic is to traverse all the nodes to find the node with the smallest remaining space and able to fit the task when assigning the task. In this experiment, space is represented by the product of different dimensions (processing unit times words). The smallest space that can be stored is selected. This strategy maximizes the use of space in some situation [16].

3. The assign_task_nf method follows the next fit allocation principle. The next-fit allocator picks up its search from where it last left off [17]. In this case, the assign_task_nf method always looks for nodes that can meet the task requirements in the nodes after the node that was last assigned the task. This strategy can distribute tasks somewhat evenly but may not utilize node resources as well as other strategies.

4. The assign_task_wf method follows the principle of worst fit allocation. This strategy always tries to find the node with the largest remaining space and assigns the task to it. This strategy can also distribute tasks somewhat evenly.

This class requires a NodeManager object at creation time, it uses NodeManager to get a list of Node objects and uses these Node objects to perform tasks. The class also uses the Node's lock property to ensure that task assignment is thread safe. This class doesn't come equipped with methods for initiating or halting nodes. Consequently, you're required to utilize methods from NodeManager for activating a node prior to employing this class for task distribution. Similarly, NodeManager's methods must be used for shutting down a node once the assignment of all tasks is fully accomplished.

- The function called 'data_generator' is constructed to produce a sequence of randomized tasks. Each task is represented by a string, assembled from randomly selected words. This function's purpose is to supply the necessary data for executing the primary program during experimental operations. The function allows control over the number of tasks to be generated through the 'num' parameter The 'data_len' parameter control the maximum word count for each task (the actual count ranges between 1 and the 'data_len' value). The 'randomseed' parameter is controled to initialize the random number generator, ensuring that each time the same seed is used, the sequence of tasks generated remains consistent.

- The Polt class is used to show the result of the allocation of the nodes. show_stat_and_plot function calculates and displays statistics about the distribution of tasks among nodes (such as the standard deviation of task count, core count, and time count per node), and how many nodes have been assigned tasks. It visualizes these statistics using a plot, making it easier to see the distribution and usage of tasks, processing units (cores), and words counts across different nodes.

- The Test class is used to examine substantial volumes of data, documenting the outcomes of each individual test within a list for reference and further analysis. In this class, some initial parameters should be set up: the maximum tasks that can be processed per node, throughput, maximum time, number of nodes, and the number of tasks. then enters a loop that runs preset times. In each iteration: It generates a certain number of tasks using the data_generator function. A new NodeManager is created with the specified parameters. A TaskScheduler is created using NodeManager. The time taken to assign tasks to nodes using the assign_task_ff function is measured. The show_stat_and_plot function is called, which prints statistics about the task assignment and shows a plot, and returns the standard deviations of task counts, core counts, and time counts per node, as well as the number of nodes used. The nodes are then started, and then closed, presumably processing the tasks. The standard deviations and number of nodes used are then added to lists for later analysis. After the loop finishes, it prints the parameters used in the test, as well as the standard deviations of task counts, core counts, time counts per node, number of nodes used, and the time it took to perform the task assignment for each iteration of the loop. The purpose of this script is likely to test and analyze the performance of different assign functions under various conditions, such as varying task loads, and compare the standard deviations of task counts, core counts, and time counts. This specific data can be harnessed to assess both the efficiency and the equity of the implemented scheduling algorithm. For example, lower standard deviations would indicate that tasks are more evenly distributed across nodes.

In the code mentioned above, NodeManager can be regarded as the "master" responsible for task distribution, and each node can be regarded as the "worker" that executes the task TaskScheduler is responsible for assigning tasks to nodes according to a certain strategy.

3.3 Approach

Allocation algorithm module is designed to solve a problem similar to a two-dimensional bin packing problem, Maximize the use of space in a two-dimensional space [18]. taking the number of words and the number of processing units as the two dimensions. The algorithm module allocates tasks of different shapes to the nodes.

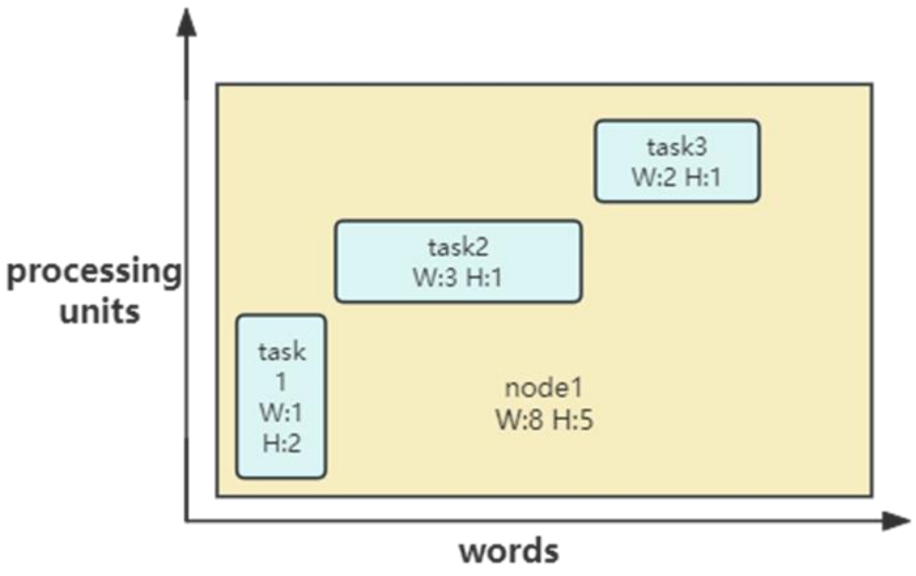


Figure 4: Bin Packing Problem instance

Taking figure 4 as an example, the yellow block can be seen as a container representing the space of a node, and the blue blocks represents tasks. The height of the yellow block represents the node's throughput, while the length of the yellow block represents the node's max words. The number of yellow blocks indicates the number of nodes. The quantity of blue blocks represents the number of tasks, the height of the blue blocks represents the number of processing units required for the task, and the length of the blue blocks signifies the number of words in the task. When allocating tasks to nodes, it's imperative to ensure that the cumulative length of all tasks remains within the node's capacity constraints. Similarly, the aggregate height of all tasks should not surpass the node's specified height limit. The same tasks and nodes can have different placement strategies, and the allocation algorithm here is equivalent to the placement strategy.

In summary, the following parameters are used to measure nodes and tasks.
The following parameters are used to evaluate the **nodes**:

- Maximum tasks: the maximum number of tasks that a node can handle (max blue blocks number in a yellow block).
- Through put: the number of processing units in a node (height of a yellow block).
- Maximum words: the total number of words a node can process (length of a yellow block).
- Number of nodes: the total count of nodes (yellow blocks number).

The following parameters are used to measure the **tasks**:

- Number of tasks: the total number of tasks in the data (blue blocks number).
- Maximum processing unit: the maximum number of processing units that a task

uses (The maximum height among all the blue blocks).
- Maximum words: the maximum number of words in a task (The maximum length among all the blue blocks).

The experimental results mainly evaluate the allocation algorithm module through three aspects:
Load balancing: Load balancing measures the even distribution of tasks or workloads across various processing nodes or resources.
Runtime: The total time taken from the start to the completion of a task.
Resource utilization: Evaluates how effectively the available resources, such as CPU, memory, disk I/O, and network, are used by the allocation algorithm.

According to the three aspects above, the following **result parameters** are used.
- Standard deviation of nodes' tasks number: the evenness of the number of tasks allocated among the nodes (Load balancing).
- Standard deviation of nodes' processing unit used: the evenness of the use of processing units among the nodes (Load balancing).
- Standard deviation of nodes' words number: the evenness of the number of words assigned among the nodes (Load balancing).
- Fit execution time: the running time of the allocation algorithm (Runtime).
- Number of tasks assigned divided by the number of nodes used: the utilization efficiency of the node (Resource utilization).

The experiment primarily employs the controlled variable method. By controlling one node or task parameter while keeping other parameters constant, four allocation algorithms were separately tested to observe the changes in result parameters and the performance differences between the different algorithms. The data required for the experiment is randomly generated by the data generator according to the set parameters. To avoid the randomness of the results of randomly generated data, we use different random seeds for testing.

4. Finding and Discussion

4.1 Results

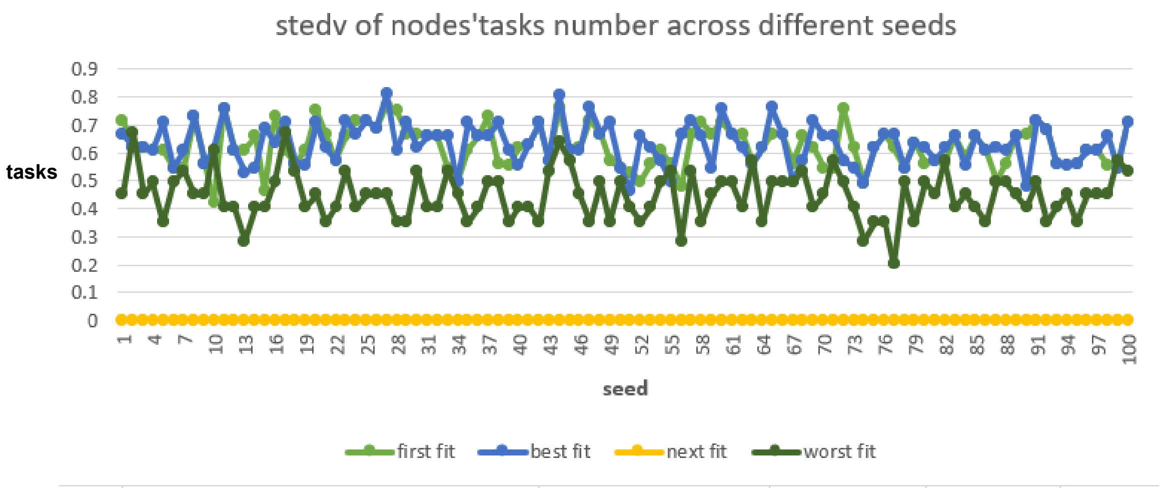
Experiment 1

In the following parameter settings, 100 tests are conducted separately for the four algorithms. The data for the 100 tests come from the results generated by the data generator with random seeds from 0 to 99.

Nodes Configuration			
Max task	Through put	Max words	Number of nodes
4	20	22	50

Task(word) Configuration		
Max processing unit	Max words	Number of tasks
9 (average≈5)	10 (average≈5.5)	100

Table 1: Input of Experiment 1



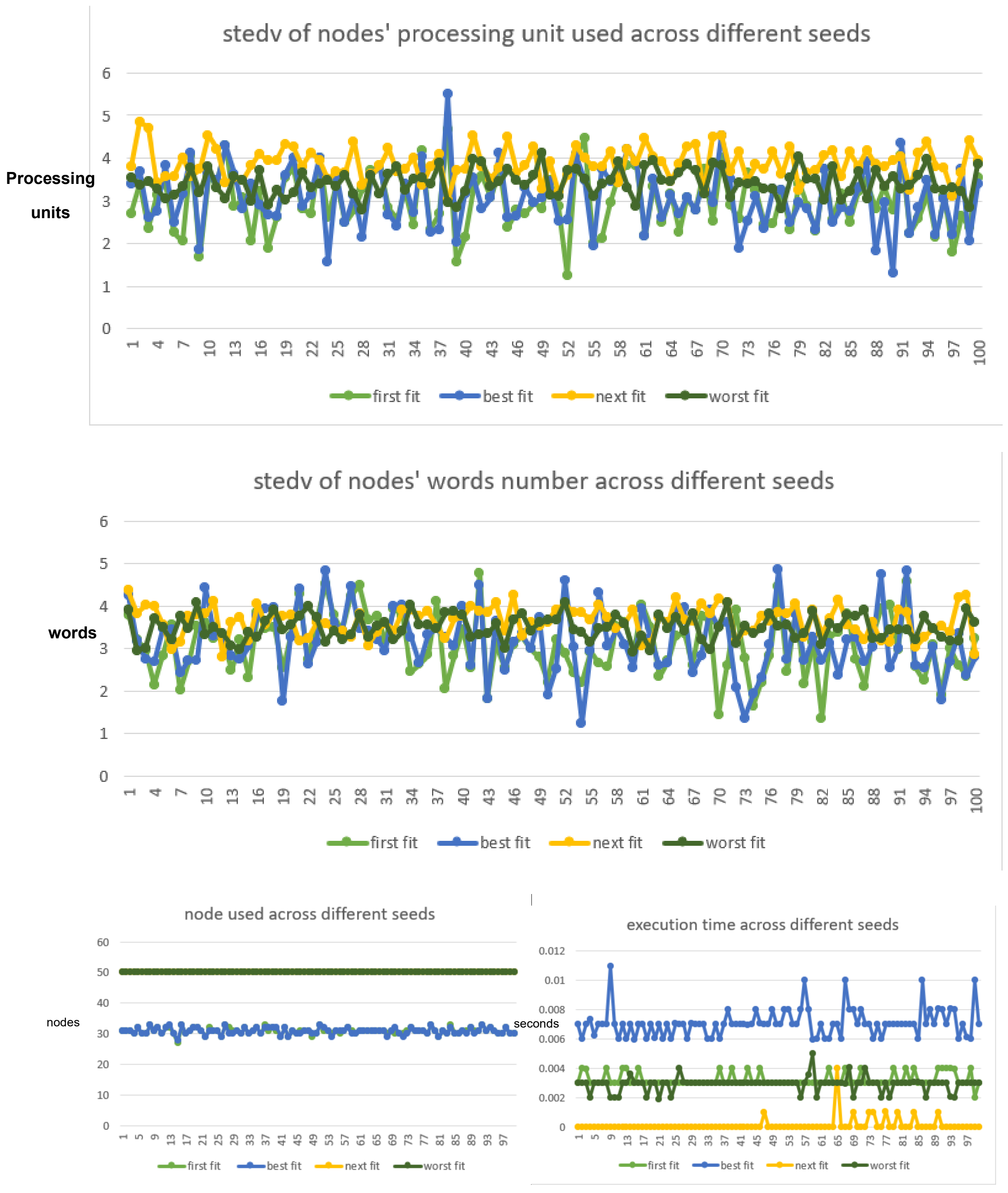


Figure 5: Results parameter of Experiment 1

According to the data and figure 5 the following observations can be made:
The results of standard deviation of nodes' tasks number indicate that First Fit and Best Fit algorithm have very close averages in 100 tests, with a difference of only 1.5%. The value for Worst Fit is significantly lower than the above two algorithms, suggesting that the number of tasks allocated to each node is more uniform with Worst Fit algorithms under these circumstances. The value for the Next Fit algorithm is 0, which means that the Next Fit algorithm is absolutely uniform in this situation. There are two

main reasons for this phenomenon. Firstly, the ample number of nodes results in a very low space utilization rate for each node, leaving plenty of space to accommodate the next task. Task allocation will be unrestricted. Secondly, the number of tasks can be exactly divided by the number of nodes in this situation. 100/50 = 2, meaning each node evenly receives two tasks. Therefore, in this example, the 'Standard deviation of nodes' tasks number' result for Next Fit algorithm doesn't have much reference value.

The results of standard deviation of nodes’ processing unit used show that Next Fit algorithm's processing unit usage is the least uniform, followed by Worst Fit algorithm. Best Fit and First Fit algorithm’s value are very similar and are more uniform than the previous two algorithm’s value.

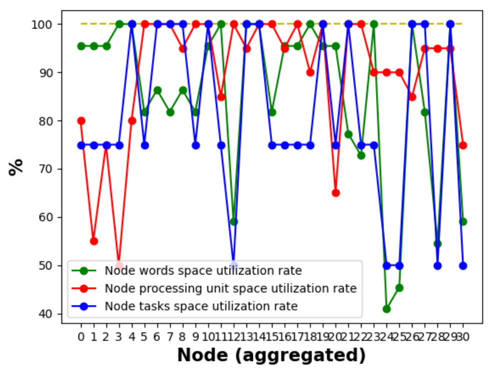
The results of standard deviation of nodes' words number show that, again, the word allocation per node is the least uniform under the Next Fit algorithm, followed by Worst Fit algorithm. The results for Best Fit and First Fit algorithm are very similar and are the most uniform compared to the previous two algorithms. However, the difference in results between the four algorithms is not very pronounced.

For 'node used number', it is easy to see that Worst Fit algorithm and Next Fit algorithm use all 50 nodes, while the number of nodes used by First Fit algorithm and Best Fit algorithm is around 31.

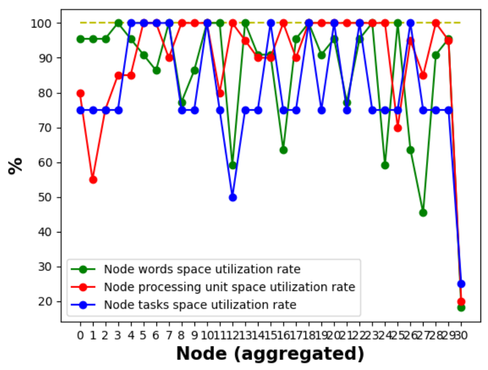
The results for 'fit execution time' show that Best Fit algorithm takes the most time, close to double that of First Fit algorithm, while the time for Worst Fit and First Fit algorithm is quite similar. Next Fit algorithm absolutely leads all other algorithms, with time close to 0.

In terms of node utilization, we use the number of tasks divided by the number of nodes to find that First Fit and Best Fit algorithm have similar node utilization rates, which are much higher than Next Fit algorithm and Worst Fit algorithm. The utilization rates for Next Fit and Worst Fit algorithm are the same.

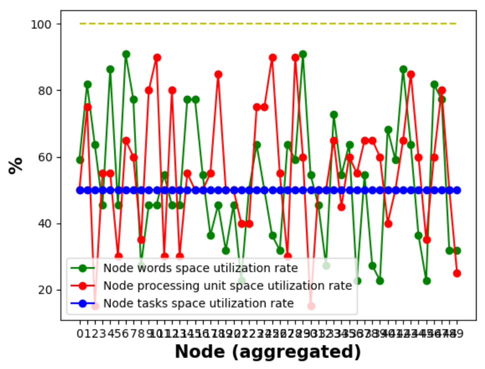
Next, using the data generated with the random seed 0 as an example, under the premise of the above parameter settings, use the plot code to show the distribution of tasks on nodes under different algorithms. It's important to note again that this node space utilization rates chart displays the utilization status of the nodes that were assigned tasks. When there are too many nodes, multiple nodes (depending on the total number of nodes) will be grouped together, and their average values will be displayed.



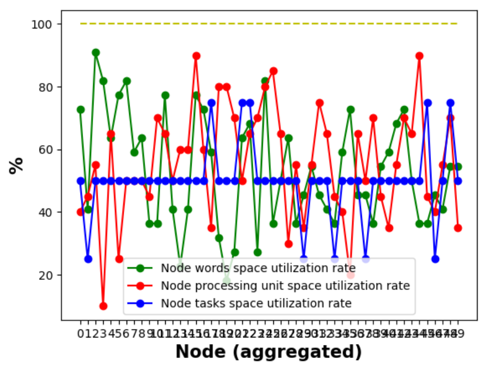
(First Fit node space utilization rates chart)



(Best Fit node space utilization rates chart)



(Next Fit node space utilization rates chart)



(Worst Fit node space utilization rates chart)

Figure 6: Node Space Utilization Rates

For the First Fit and Best Fit, the utilization of each parameter of the nodes is very high and about between 70 and 100. However, the node utilization under the Next Fit and Worst Fit algorithms is insufficient. This is because the principle of the First Fit and Best Fit algorithms is to use as few nodes as possible, whereas the principle of the Next Fit and Worst Fit algorithms is to assign tasks if there are nodes available.

Next, keeping other parameters constant, we proportionally increase the number of tasks and nodes. The purpose of this is to verify whether increasing the data volume under unchanged node and task attributes will lead to different results.

Tests have been conducted by increasing the number of nodes and tasks for experiment 2 (250, 500) and experiment 3 (500, 1000).

Experiment 2

Nodes Configuration			
Max task	Through put	Max words	Number of nodes
4	20	22	250

Task(word) Configuration		
Max processing unit	Max words	Number of tasks

9 (average≈5)	10 (average≈5.5)	500

Table 2: Input of Experiment 2

Experiment 3

Nodes Configuration			
Max task	Through put	Max words	Number of nodes
4	20	22	500

Task(word) Configuration		
Max processing unit	Max words	Number of tasks
9 (average≈5)	10 (average≈5.5)	1000

Table 3: Input of Experiment 3

Comparing and observing experiment 1, 2, and 3, it can be found that the ranking of the result parameters does not change as the data volume increases, and the numerical changes are not significant. Interestingly, as the data volume expands, the numerical stability of the result parameters across different seeds (reflected in smaller fluctuations in the color lines on the chart), indicating more stable and reliable results. Additionally, the differences between different algorithms remain consistent with experiment 1, but the gaps between different algorithms become clearer. We can draw a preliminary conclusion that only increasing the data volume (proportional increase of task quantity and node quantity) will not affect the relative performance of different algorithms. Instead, a larger volume of data represents more reliable and stable results.

To investigate whether the performance of different algorithms under massive data would differ from experiment 1, the task quantity is increased to 10,000 and the node quantity to 5,000 for experiment 4. From the above conclusion, we know that when the data volume is large, the results tend to be consistent across different seeds. Therefore, for the sake of program execution convenience, the experiment is simplified, and testing will be conducted using 10 seeds.

Experiment 4

Nodes Configuration			
Max task	Through put	Max words	Number of nodes
4	20	22	5000

Task(word) Configuration		
Max processing unit	Max words	Number of tasks
9 (average≈5)	10 (average≈5.5)	10000

Table 4: Input of Experiment 4

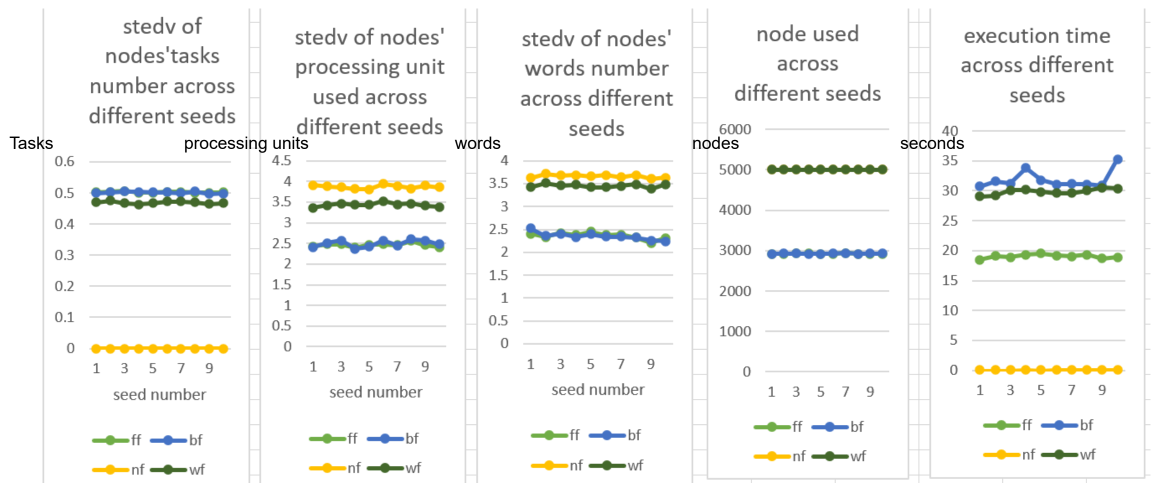
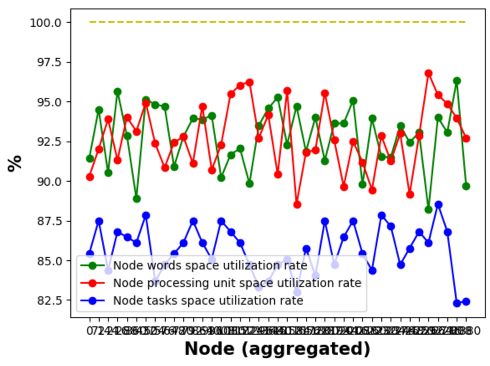
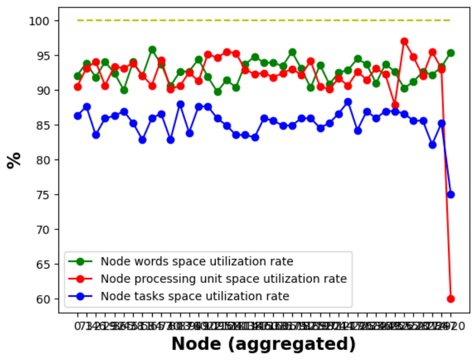


Figure 7: Results parameter of Experiment 4

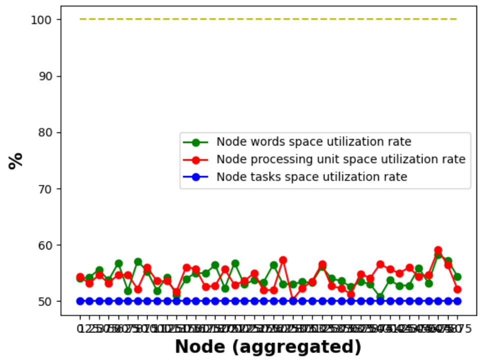
As seen in the graphs, when the task quantity is large enough, the results indeed match the previous predictions, showing greater consistency across different seeds. A point worth noting is that as the task volume increases, the execution time of the Worst Fit algorithm increases significantly, approaching that of the Best Fit algorithm. The reason might be that with fewer tasks, the Worst Fit algorithm can quickly find the smallest space, similar to the First Fit algorithm, but the time complexity of the Worst Fit algorithm is similar to that of the Best Fit algorithm, so it approaches Best Fit algorithm with a large number of tasks.



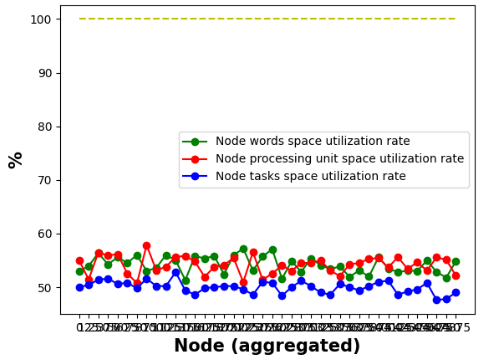
(First Fit node space utilization rates chart)



(Best Fit node space utilization rates chart)



(Next Fit node space utilization rates chart)



(Worst Fit node space utilization rates chart)

Figure 8: Node Space Utilization Rates

By observing the node space utilization rates chart of seed 0 in experiment 4, it can be found that the utilization rate of each parameter basically remains consistent with experiment 1. As in experiment 1, when a sufficient number of nodes is set, the word number and processing unit need of Next Fit and Worst Fit algorithms are far from reaching the maximum capacity of the nodes.

To further explore the differences when Next Fit and Worst Fit algorithms have a high utilization rate, we need to examine the impact on these two algorithms when the number of nodes is insufficient. Because in experiment 4, the number of nodes used by First Fit and Best Fit algorithms is close to 3000. Make the number of nodes reduced to 3000 in experiment 5. Since the node usage rate of Next Fit and Worst Fit algorithm is lower, there will be results with unassigned tasks. In this case, Number of unassigned tasks is added as a new indicator.

Experiment 5

Nodes Configuration			
Max task	Through put	Max words	Number of nodes
4	20	22	3000

Task(word) Configuration		
Max processing unit	Max words	Number of tasks
9 (average≈5)	10 (average≈5.5)	10000

Table 5: Input of Experiment 5

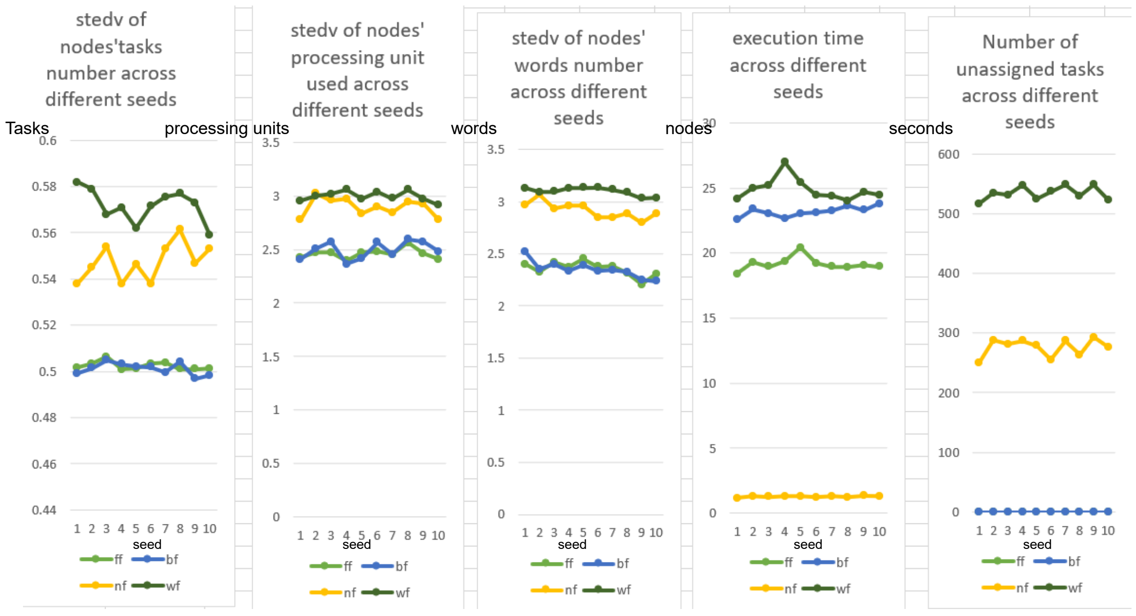
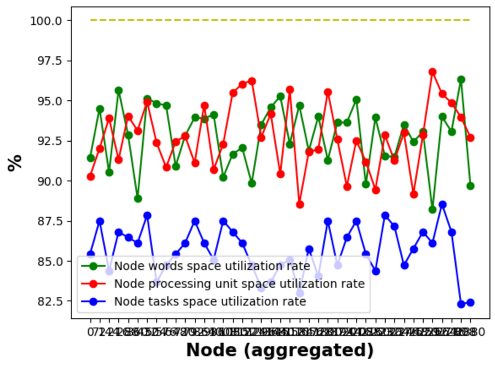


Figure 9: Results parameter of Experiment 5

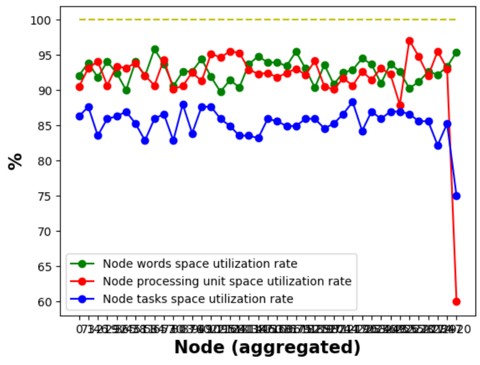
As depicted in Figure 9, the Worst Fit algorithm consistently exhibits the highest values across all evaluated parameters, followed by Next Fit algorithm. The values of First Fit and Best Fit algorithm are still similar and less than the previous two algorithms. What's different from before is that Worst Fit 's standard deviation of nodes' tasks number, standard deviation of processing unit used, standard deviation of words' number become the highest. When the number of nodes is insufficient (i.e., the node utilization rate is high), the performance of the Worst Fit algorithm clearly declines. This happens because the Worst Fit algorithm tries to distribute the tasks evenly across all available nodes. When the number of nodes is limited, it struggles to find a suitable node for each task, resulting in unassigned tasks and lower overall performance.

In terms of the number of nodes used, all the algorithms used about 3000 nodes. In terms of execution time, Worst Fit algorithm is the highest, followed by Best Fit algorithm, then First Fit algorithm, and Next Fit algorithm still maintains excellent speed.

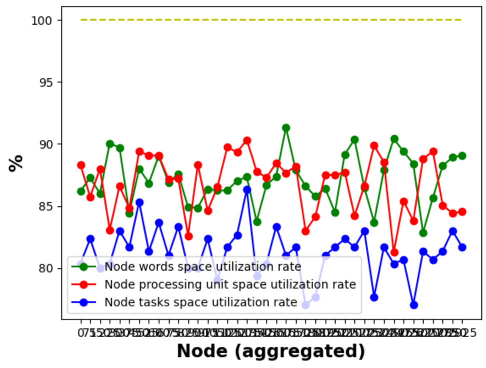
The result of Number of unassigned tasks shows that First Fit and Best Fit algorithm have no unassigned tasks. Next Fit algorithm has an average of 276 unassigned tasks, and Worst Fit algorithm has an average of 534 unassigned tasks. When the number of nodes is consistent, the fewer tasks assigned, the lower the utilization rate of the nodes. The utilization rate of Next Fit is lower than that of First Fit and Best Fit algorithm, and the node utilization rate of Worst Fit algorithm is lower than that of Next Fit algorithm.



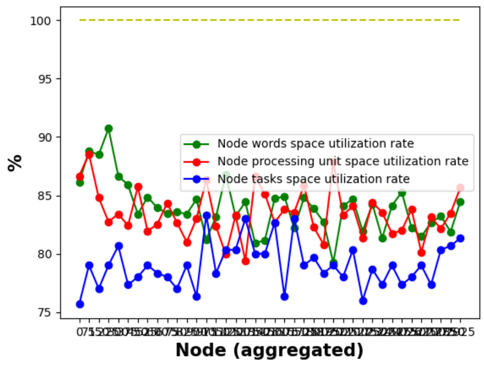
(First Fit node space utilization rates chart)



(Best Fit node space utilization rates chart)



(Next Fit node space utilization rates chart)



(Worst Fit node space utilization rates chart)

Figure 10: Node Space Utilization Rates

According to the node space utilization rates chart, the distribution of First Fit and Best Fit algorithm is the same with experiment 4. Next Fit and Worst Fit algorithms have a significant increase in utilization. However, Next Fit ang Worst Fit face tasks unassigned problem.

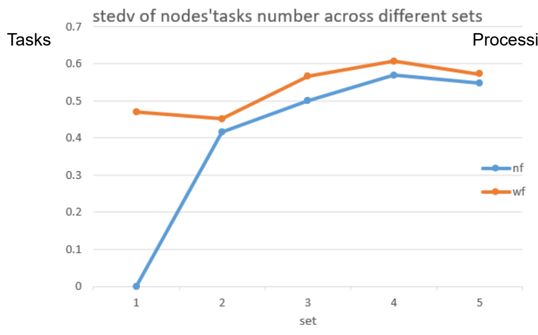
Experiment 6

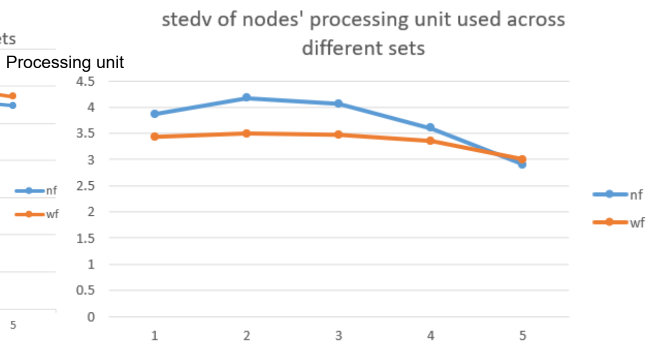
Nodes Configuration			
Max task	Through put	Max words	Number of nodes
4	20	22	5k,4.5k,4k,3.5k,3k

Task(word) Configuration		
Max processing unit	Max words	Number of tasks
9 (average≈5)	10 (average≈5.5)	10000

Table 6: Input of Experiment 6

Experiment 6 maintain all other parameters and gradually reduce the number of nodes from 5000 to 3000, using 4500, 4000, and 3500 as transitions to study the performance of each algorithm module when the number of nodes is reduced. As known from the algorithm principles and result from experiment 4 and 5, the First Fit and Best Fit algorithm modules will not cause changes in values when the number of nodes is greater than the number of tasks to be assigned, only negligible differences in execution time will occur (the First Fit and Best Fit algorithms will not bring any changes to the results when the number of nodes is greater than the minimum required number of nodes, the results remain the same). The smallest number of test nodes in experiment 6 is 3000, which is greater than the number of nodes required by the First Fit and Best Fit algorithm modules in experiment 4. Therefore, only test the Next Fit and Worst Fit here. The test compares different sets with different number of nodes: set1 with 5000 nodes, set2 with 4500 nodes, set3 with 4000 nodes, set4 with 3500 nodes, and set5 with 3000 nodes.





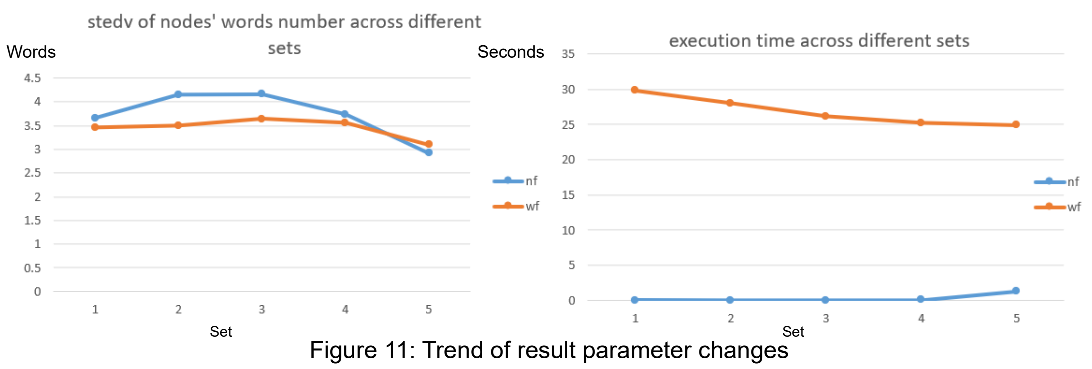


Figure 11: Trend of result parameter changes

From figure 11, it can be observed that the Next Fit and Worst Fit algorithm modules show a trend of rising first and then falling in the first three graphs. The reason might be that the reduction in the number of nodes limits the uniformity of Next Fit and Worst Fit, causing an increase (indicating unevenness). When the number of nodes is reduced to a certain extent, some tasks cannot be assigned, which leads to a drop in indicators (because some task assignments are skipped, it becomes more uniform). The performance on execution time does not differ much. Compared with the First Fit and Best Fit algorithm modules, the results are basically consistent with previous tests.

experiment 6 shows that reducing the number of nodes for Next Fit and Worst Fit algorithms will increase restrictions, leading to a decline in the uniformity of each node allocation, and even a phenomenon where there are not enough node spaces for task allocation. Next Fit and Worst Fit algorithms are more suitable for running when there are sufficient nodes. When there are sufficient nodes, Next Fit has a leading speed advantage, but the uniformity of Worst Fit in various parameters is higher than Next Fit.

All the above experiments are conducted under the condition that the usage in each parameter is similar (the maximum space of each parameter of the node is about 4 times the average task demand). Consider a scenario in which the maximum allowable space of a particular node parameter is reduced (or the average demand of a given task parameter is increased). This leads to the question of the potential impacts on each algorithmic module. Given that the processing unit and the word count function independently and occupy interchangeable positions, the processing unit is selected as the focus of this study. By decreasing the node throughput and imposing a stricter constraint on the processing unit, this investigation seeks to ascertain alterations in the performance of each algorithm.

Experiment 7

SET1

Nodes Configuration			
Max task	Through put	Max words	Number of nodes
4	20	22	5000

Task(word) Configuration		
Max processing unit	Max words	Number of tasks
9 (average≈5)	10 (average≈5.5)	10000

SET2

Nodes Configuration			
Max task	Through put	Max words	Number of nodes
4	18	22	5000

Task(word) Configuration		
Max processing unit	Max words	Number of tasks
9 (average≈5)	10 (average≈5.5)	10000

SET3

Nodes Configuration			
Max task	Through put	Max words	Number of nodes
4	16	22	5000

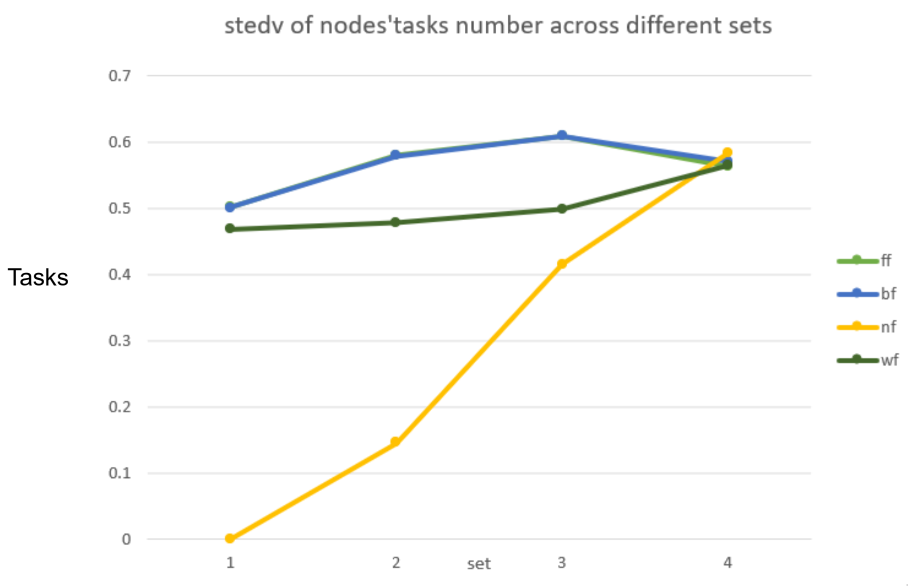
Task(word) Configuration		
Max processing unit	Max words	Number of tasks
9 (average≈5)	10 (average≈5.5)	10000

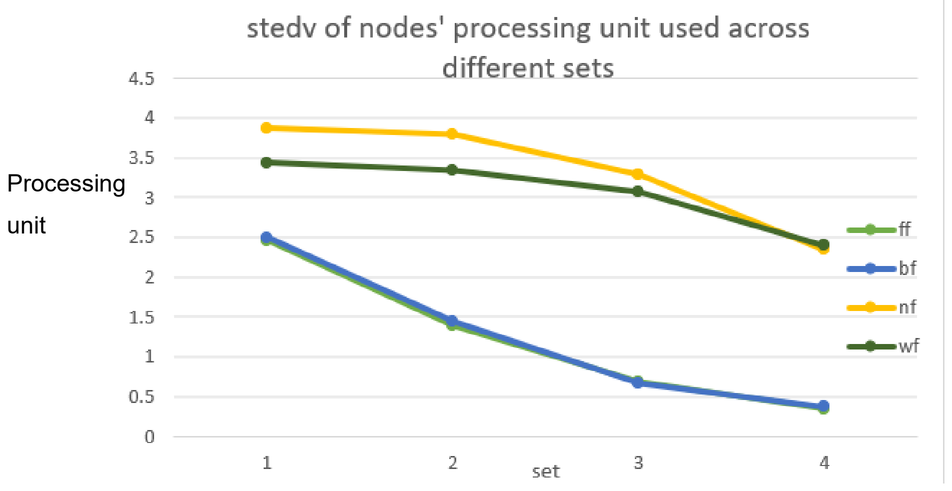
SET4

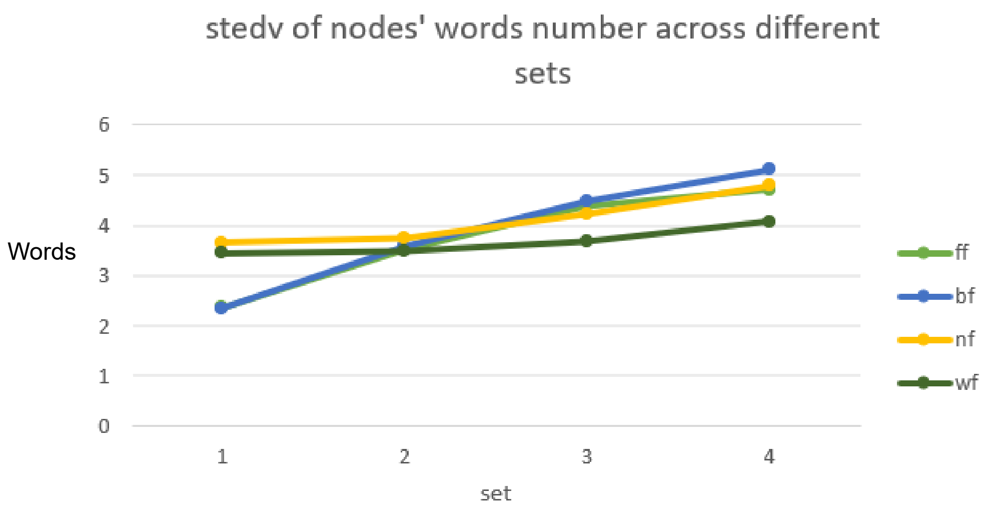
Nodes Configuration			
Max task	Through put	Max words	Number of nodes
4	14	22	5000

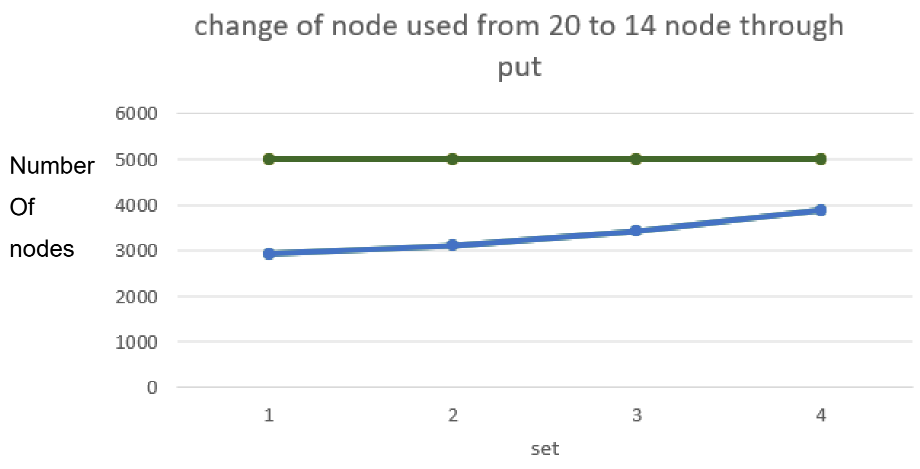
Task(word) Configuration		
Max processing unit	Max words	Number of tasks
9 (average≈5)	10 (average≈5.5)	10000

Table 7: Input of Experiment 7


stedv of nodes'tasks number across different sets
Tasks
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
set
3
4
ff
bf
nf
wf


stedv of nodes' processing unit used across different sets
Processing unit
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
1
2
set
3
4
ff
bf
nf
wf


stedv of nodes' words number across different sets
Words
6
5
4
3
2
1
0
1
2
3
4
set
ff
bf
nf
wf



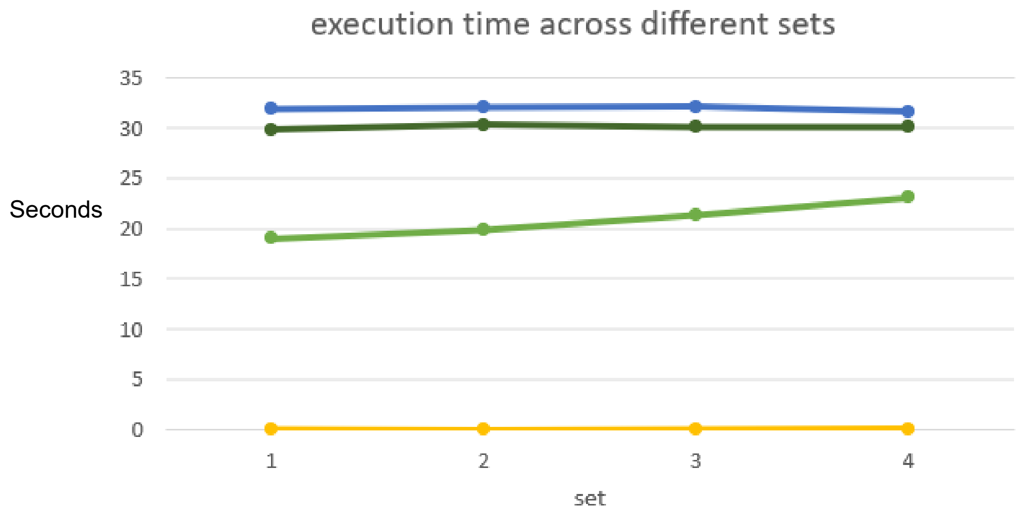


Figure 12: Trend of result parameter changes

Under these conditions, the throughput parameter is designated as the "constrained parameter." This nomenclature arises from the fact that while the maximum values of other parameters remain constant, the throughput value exhibits a progressive decline from set1 through to set4.

Based on the figure 12 representation, a discernible pattern emerges with the decreasing throughput: the standard deviation of nodes' tasks number to nodes for both the First Fit and Best Fit algorithms initially increases, followed by a subtle decline, indicating a predominant upward trajectory. The Worst Fit shows a slow rising trend, while the Next Fit algorithm increases significantly.

The standard deviation of processing unit used of the four algorithm modules all show a downward trend. This indicates that the main limiting parameter under these algorithms will make the distribution of this parameter more uniform. It can be clearly seen that the values of First Fit and Best Fit algorithms are lower than those of Next

Fit and Worst Fit algorithm modules.

The standard deviation of nodes' words number chart shows that all algorithms show an upward trend on this indicator. It's worth noting that here, when the throughput value is equal to 14, the Best Fit algorithm's indicator is higher than First Fit algorithm's, which is the biggest difference manifested between First Fit and Best Fit algorithm so far in the tests.

The number of nodes in Next Fit and Worst Fit algorithm remains unchanged at 5000, but when through put is reduced to 14, there will be a phenomenon of insufficient node space and inability to allocate tasks. The number of tasks Worst Fit cannot allocate is greater than Next Fit algorithm. The number of nodes in First Fit and Best Fit algorithms increases as throughput decreases. The node utilization rate still maintains that First Fit equals Best Fit algorithm, greater than Next Fit, which is greater than Worst Fit. There is not much change in execution time. Best Fit takes the longest time, followed by Worst Fit, then First Fit, and Next Fit takes the least time.

It can also be postulated that, given a decrease in the node's maximum word value while other parameters remain constant, the outcomes would align closely with those observed in Experiment 7. Since the principles of the several algorithms participating in the experiment treats different dimensions without priority setting, in the results, the number of words and the number of processing units have the same status.

As throughput diminishes, it becomes evident that the space utilization rates in dimensions exclusive of throughput are notably subdued. At this point, the primary determining factor as to whether there are sufficient nodes becomes the throughput.

To validate that increasing task size and decreasing the upper limit of node capacity will have the same effect on different algorithms as observed in experiment 7, experiment 8 was designed in parallel to experiment 7. To simplify the experiment, two sets were set up for testing.

Test 8

SET1

Nodes Configuration			
Max task	Through put	Max words	Number of nodes
4	20	22	5000

Task(word) Configuration		
Max processing unit	Max words	Number of tasks
9 (average≈5)	10 (average≈5.5)	10000

SET2

Nodes Configuration

Max task	Through put	Max words	Number of nodes
4	20	22	5000

Task(word) Configuration		
Max processing unit	Max words	Number of tasks
9 (average≈5)	14 (average≈7.5)	10000

Table 8: Input of Experiment 8

From the provided dataset, it becomes evident that "words" serve as the constrained parameter in this scenario. The change in the uniformity of task allocation is the same as in Test 7, manifested as an increase in the stander deviation values of other dimensions and a decrease in the stander deviation of "words". This essentially verifies that the effects of reducing the node limit and increasing the task size are the same.

4.2 Summary

First Fit and Best Fit Algorithms:

• High node utilization rates and uniform distribution across tasks, processing units, and words.

• These algorithms take longer to run and are unable to distribute tasks across a specific number of nodes.

• Best Fit algorithm takes longer than the First Fit algorithm.

• The uniformity of the Best Fit algorithm is worse than that of the First Fit algorithm when nodes are insufficient.

Next Fit and Worst Fit Algorithms:

• These algorithms distribute tasks across a specific number of nodes.

• Next Fit algorithm has an advantage in speed and can quickly allocate all tasks.

• The Worst Fit algorithm doesn't have the speed of the Next Fit algorithm, but its resource distribution is more uniform.

• If the number of nodes is not sufficient, both algorithms will have tasks that can't be allocated and the uniformity of task distribution decreases.

4.3 Limitations

The design and implementation of experiments often come with many limitations and challenges. In this study, perhaps the most significant drawback is that the entire experiment was conducted in a simulated environment, using entirely virtual data. While such a setup provides a means to control experimental variables, it presents a clear concern: performance in a simulated setting may not accurately reflect the actual performance in a real, complex environment. It is worth noting that the actual MapReduce framework may involve more complex issues such as communication between nodes, fault recovery, data location optimization, etc.

A genuine operational environment is far more intricate than a simulated one, involving a greater number of parameters. The slightest variation in any parameter could have profound effects on system performance. In designing the experiment, we might not have taken into account all potential parameters and variables, introducing certain limitations to our research. Although our findings appear valid within the simulated context, they may face distinct challenges and issues in real-world applications.

What worth to be mentioning that even though the Best Fit (BF) algorithm employed a more sophisticated strategy than the First Fit (FF) during the experimental design phase, the eventual results indicated that BF's performance was even lower to FF. This serves as a reminder that an algorithm's complexity does not always correlate positively with its performance – sometimes, simpler strategies might be more efficient.

Nevertheless, a significant contribution of the experiment is our successful modularization of the allocation algorithm within a distributed framework, further metricizing these modules. This lays a solid foundation for subsequent modular adaptive studies. On the other hand, we only tested modularization in a part of the MapReduce framework, failing to showcase the module's full performance in the overall system. This suggests that, while we've made some intriguing initial findings, a comprehensive assessment of these modules' impact and performance across the system requires further experimentation and research.

In conclusion, while our research has shortcomings in certain areas, it also offers direction and insights for future studies. With continued effort and exploration, we hope we can gain a deeper understanding of task scheduling algorithms in distributed computing environments and provide more scientific and logical solutions for real-world applications.

5. Future

More allocation algorithms:
Previous research has extensively analyzed algorithms such as First Fit, Best Fit, Next Fit, and Worst Fit. Besides these algorithms we have studied, there are many other algorithms such as Round Robin, Priority-based, Fair Share, etc. These algorithms may have performance beyond the previous four algorithms in specific scenarios. When task priority is a concern, for instance, the Priority-based algorithm may be a better option. It is because each algorithm has its unique advantages and limitations that we plan to further extend our research by introducing more types of allocation algorithm modules.

Overall, our goal is to provide the most suitable and optimal solutions for different practical application scenarios based on a more comprehensive understanding of various task scheduling algorithms.

More data types:
Data, as the core of modern computing, has a crucial impact on the performance of task scheduling algorithms due to its diversity and complexity Based only on the characteristics of the input data, a specific strategy may succeed in one situation and fail in another. This underscores the importance of including diverse data types. Different data types often require different processing and analysis strategies. For example, in the case of text data, considerations might include its length, intricacy, and pertinence. Meanwhile, for image data, factors such as size, clarity, and content complexity play a role. Grasping these data attributes allows us to refine and boost the efficiency of our algorithms more effectively. We will add a variety of data types for testing and increase the parameters of data types. This will make the allocation algorithm more targeted, allowing different algorithm types to be chosen for different data types.

A more refined framework:
Although the modular testing of the framework has been partially realized in the current study, there are still many parts waiting for us to explore and improve. The current simulation framework already contains some basic modules and functions, but the real MapReduce environment involves much more problems and challenges than these. For instance, ensuring task fault tolerance, slicing, and dicing data effectively, and optimizing data localization are all significant directions that merit additional study. The true issue is in putting these modules and functions together into an effective and reliable system. Testing a single module or function can help us understand its fundamental performance and limits. This requires us to deeply consider the interactions between modules to ensure overall synergy. A high-quality simulation framework should be as close as possible to the actual production environment, thus providing us with real and accurate performance data and feedback. We hope to

improve the simulation framework so that it can better simulate the various situations and challenges in real environments. Data processing requirements and technologies are constantly evolving. A successful simulation framework should not only meet current needs, but also be scalable and adaptable to future developments and changes. We hope to make the simulation framework more flexible and scalable through modular design and development. We will continue to enrich the simulation environment and improve the MapReduce framework to replicate real-world scenarios as closely as possible. Additionally, we will gradually add new modules.

In summary, improving the MapReduce simulation framework is not only to improve the depth and breadth of our research, but also to ensure that our research results can be truly applied to real production environments and provide effective support and solutions for big data processing.

AI technology:
In our research, we believe that AI technology can greatly accelerate and optimize the module calling and assembly process of the MapReduce simulation framework. Traditional module assembly and selection usually requires researchers to proceed based on experience and intuition. However, as the number and complexity of modules increase, this approach becomes less and less feasible.AI can automatically select the most appropriate modules for a given application and requirement by learning from past data and patterns and assemble them into an optimal system structure. Neural networks may be taught to recognize which module combinations perform best for a specific job or sort of data.

We will introduce AI technology to greatly improve the efficiency and effectiveness of module assembly and achieve true intelligence and automation. This will not only bring more possibilities and opportunities for our research, but also provide a more robust and stable solution for the actual production environment.

6. Conclusion

From the in-depth tests conducted within our simulated MapReduce framework, the following significant insights have been gleaned:

Characteristics of First Fit and Best Fit:
Through the constructed simulation framework testing, we can find that the First Fit and Best Fit algorithms have high node utilization rates and use the fewest nodes, with the most uniform distribution in assigned node across three parameters (tasks, processing unit, words). The disadvantage is that they take longer to run, and unable to evenly distribute tasks across a specific number of nodes.

In the current experiments, not only does the Best Fit algorithm take longer than the First Fit algorithm, but the uniformity of the Best Fit algorithm is even worse than that of the First Fit algorithm when nodes are insufficient. If only the parameters in the experiment are referred to, the First Fit algorithm can replace the Best Fit algorithm as a better choice. This result goes beyond the previous expectation, as the Best Fit algorithm uses a more complex logical structure. But this is the purpose of the experiment, sometimes the results will not be as expected, and more complex is not necessarily better. At least in the above data test, the data of the First Fit algorithm is superior to or equal to the Best Fit algorithm.

Characteristics of Next Fit and Worst Fit:
Unlike the first two algorithms, the Next Fit and Worst Fit algorithms will distribute tasks across a specific number of nodes unless the number of tasks is less than the number of nodes. The Next Fit algorithm has an absolute advantage in speed which can quickly allocate all tasks. However, the allocation of resources of nodes is not uniform. Although the Worst Fit algorithm does not have the speed of the Next Fit algorithm, its distribution of resources is more uniform than that of the Next Fit algorithm. It should be noted that if the number of nodes is not sufficient, it will affect the effect of the Next Fit and Worst Fit algorithms, both the Next Fit and Worst Fit algorithms will have tasks that cannot be allocated, and the uniformity of task distribution also decreases. The changes in node parameters and task parameters have less impact on the differences between various algorithms. Node parameters and task parameters mainly determine whether the number of nodes is sufficient and which dimension is restricted (in the node usage situation chart, it is the dimension closest to the real line and the dashed line). Since the dimensions in the experiment are independent of each other and do not affect each other, the algorithm does not prioritize different dimensions. It can be found that whether the node is sufficient is determined by the dimension with the greatest constraint.

Impact of Parameters on Algorithmic Performance:
Node and task parameters play a pivotal role in deciphering whether the node

availability suffices and which dimension faces constraints. With each dimension in the test acting independently and without mutual influence, algorithms remain impartial, not favoring any dimension. It’s paramount to discern that the adequacy of nodes is governed by the dimension facing the highest constraints.

In synthesis, choosing an optimal allocation algorithm is an exercise in meticulous consideration, necessitating decisions based on node sufficiency, preference between uniform distribution or minimal node distribution, and weighing between speed and node economy. The crux of these experimental tests is to underline the imperative of algorithmic choice, tailored to specific system requirements. The experiments shed light on the distinct advantages, limitations, and idiosyncrasies of each algorithm, facilitating an informed module selection predicated on foundational settings.

7. Reflection

The experience of this experiment gave me a deeper understanding of the viewpoint that "practice is the only criterion for testing truth". In fact, I have been obsessed with the theoretical knowledge in books and papers, thinking that mastering them means that I can perfectly apply them in practice.
When the theory was tested in a real environment, I discovered many unprecedented problems. Even a little change in a parameter will have unexpected effects. This helped me realize that real knowledge comes from both practical research and experience, in addition to books. Practical experience is like a unique compass, guiding me to find direction in the complex experimental ocean.
In fact, during the experimental planning stage, I encountered unexpected problems. I made several rookie blunders in experimental design and occasionally had to start over due to my lack of a thorough and in-depth mastery of the pertinent knowledge. This not only wastes a significant amount of time, but also makes the entire experimental process more challenging. This experience taught me an essential lesson: thorough planning and comprehension of pertinent background information are required before starting any experiment or research. Additionally, carrying out a risk assessment beforehand is essential since it can help us foresee and steer clear of any issues as early as possible, ensuring the experiment's smooth progress.

I encountered a new difficulty when I started gathering and evaluating data in the middle of the experiment. Efficiently understanding large data sets is a significant obstacle. I first tried traditional data methods. These techniques weren't sufficient when dealing with a vast volume of data. Eventually, I turned to advanced analysis methods, like Python's data tools and Matplotlib for visualization. This approach helped me assess vast data and reach precise, valuable conclusions.

In addition to learning professional knowledge such as the principles of MapReduce framework, modularity knowledge, and Python programming techniques, this experiment also taught me a methodology for scientific research. From literature research, background knowledge learning, to experimental design, execution, and analysis, every step needs to be carefully and seriously treated.
In short, although the experience of this experiment was full of challenges, it opened a new world for me, made me realize the true meaning of the path of scientific research, and also exercised my practical skills. I believe that in future research, I will face various problems more calmly and confidently, continuously exploring and making progress.

Reference

[1] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.
[2] Karloff, H., Suri, S., & Vassilvitskii, S. (2010, January). A model of computation for mapreduce. In Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms (pp. 938-948). Society for Industrial and Applied Mathematics.
[3] Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters.
[4] Cohen, S. Yahoo! partners with four top universities to advance cloud computing systems and applications research. 2013-02-25]. http://research. yahoo. com/news/2743.
[5] Hadoop wiki - powered by.http://wiki.apache.org/hadoop/Powered
[6] Langlois, R. N. (2002). Modularity in technology and organization. Journal of economic behavior & organization, 49(1), 19-37.
[7] Gannon, J. D., Hamlet, R. G., & Mills, H. D. (1987). Theory of modules. IEEE Transactions on Software Engineering, (7), 820-829.
[8] Kang, W., Kapitanova, K., & Son, S. H. (2012). RDDS: A real-time data distribution service for cyber-physical systems. IEEE Transactions on Industrial Informatics, 8(2), 393-405.
[9] Hashem, I. A. T., Anuar, N. B., Gani, A., Yaqoob, I., Xia, F., & Khan, S. U. (2016). MapReduce: Review and open challenges. Scientometrics, 109, 389-422.
[10] Li, F., Ooi, B. C., Özsu, M. T., & Wu, S. (2014). Distributed data management using MapReduce. ACM Computing Surveys (CSUR), 46(3), 1-42.
[11] Coffman, E. G., Garey, M. R., & Johnson, D. S. (1984). Approximation algorithms for bin-packing—an updated survey. Algorithm design for computer system design, 49-106.
[12] Martello, S., & Toth, P. (1990). Lower bounds and reduction procedures for the bin packing problem. Discrete applied mathematics, 28(1), 59-70.
[13] Seiden, S. S. (2002). On the online bin packing problem. Journal of the ACM (JACM), 49(5), 640-671.
[14] Kang, J., & Park, S. (2003). Algorithms for the variable sized bin packing problem. European Journal of Operational Research, 147(2), 365-372.
[15] Dósa, G., & Sgall, J. (2013). First Fit bin packing: A tight analysis. In 30th International symposium on theoretical aspects of computer science (STACS 2013). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
[16] Bays, C. (1977). A comparison of next-fit, first-fit, and best-fit. Communications of the ACM, 20(3), 191-192.
[17] Kenyon, C. (1995). Best-fit bin-packing with random order (Doctoral dissertation, Laboratoire de l'informatique du parallélisme).
[18] Berkey, J. O., & Wang, P. Y. (1987). Two-dimensional finite bin-packing algorithms. Journal of the operational research society, 38, 423-429.

Appendices

Code link: practice/MAPREDUCE at master · MEMEDAbobo/practice (github.com)

Experiment data link: practice/mapreduce.xlsx at master · MEMEDAbobo/practice (github.com)