Data Collection and Gathering Data

- As mentioned, the dataset under analysis has been taken from the Kaggle website and contains multidimensional information. This information can be summarized into the following domains:

1-Cloud Computing and Cloudsim
2-Job and Task Scheduling and Planning
3- CPU Usage, Memory Usage Information and Resource Consumption Data
4-Planning Decisions
5-Job Submission and Task Dispatching

- In the following, all the columns of the dataset will be explained.

1-time
2-instance_events_type
3-collection_id
4-scheduling_class
5-collection_type
6-priority
7-alloc_collection_id
8-instance_index
9-machine_id
10-resource_request
11-constraint
12-collections_events_type
14-collection_name
15-collection_logical_name
16-start_after_collection_ids
17-vertical_scaling
18-scheduler
19-start_time
20-end_time
21-average_usage
22-maximum_usage
23-random_sample_usage
24-assigned_memory
25-page_cache_memory
26-cycles_per_instruction
27-memory_accesses_per_instruction
28-sample_rate
29-cpu_usage_distribution
30-tail_cpu_usage_distribution
31-event
32-failed

Data Pre-processing

- The initial dataset consists of 34 columns and 405,894 records or samples.

```
df.shape = (405894, 34)
```

- Here, we intend to fully explain the characteristics of the columns, and based on the descriptions, the preprocessing operations specific to each column will be mentioned:
- The first two columns were practically equivalent to row counting and contained numbers from 1 to 405,894, which were filled in ascending order.

- Therefore, we actually have 32 main columns, and we will drop and remove the first two columns.

- Based on the detailed explanations provided in the dataset report, here we will try to give a brief explanation of each of the 32 columns that you can see on the previous page. The aim is to further analyze the dataset structure so that we can gain significant insights from the raw data at our disposal. We will identify the features, and ultimately proceed with understanding, visualization, and implementing the multi-core fuzzy clustering model.

- **1-time:**
  Each record has a timer that shows the time in microseconds, calculated from 600 seconds before the start of the tracking period. This time is stored as a 64-bit integer. Additionally, there are specific time values that indicate events occurring before or after the tracking period.

- **2-instance_events_type:**
  This feature categorizes and specifies the types of events that occur within the instances, providing an analysis of the nature of operations or changes that have occurred in the tracked instances.

- **3-collection_id:**
  This is an identifier assigned to a collection, which may be a group of instances, tasks, or resource allocations. It helps to identify and reference specific collections within the dataset.

- **4-scheduling_class:**
  This feature describes the scheduling class or category for tasks or jobs, indicating their importance in the scheduling and planning system.

- **5-collection_type:**
  This feature specifies the type or category of a collection, which may distinguish between different types of collections such as job collections, machines, or resource allocations.

- **6-priority:**
  This refers to the relative priority or urgency assigned to a task or job within the scheduling system, impacting the scheduling and resource allocation.

- **7-alloc_collection_id:**
  Similar to a collection identifier, but specifically related to allocation collections, representing groups of resource reservations or allocations.

- **8-instance_index:**
  This feature indicates the index or position of an instance within a collection, used to identify and reference specific instances.

- **9-machine_id:**
  This is an identifier assigned to individual machines within the system, aiding in the tracking and management of resources at the machine level.

- **10-resource_request:**
  This feature describes the resource requirements or specifications for a task or job, indicating the resources needed for execution. The brief explanation provided is insufficient, and we will further examine this feature in detail.

- **11-constraint:**
  Constraints refer to any limitations or conditions imposed on resource allocation or task execution, ensuring compatibility with the system's process and performance.

- **12-collections_events_type:**
  Similar to the type of instance events, but related to events occurring at the collection level, providing an analysis of changes or operations affecting entire collections.

- **13-user:**
  Represents the user or entity responsible for submitting or initiating a task or job within the system.

- **14-collection_name:**
  This is the name assigned to a collection, providing a readable identifier for referencing and managing it.

- **15-collection_logical_name:**
  Represents the logical or symbolic name associated with a collection, which may be used for organizational or classification purposes.

- **16-start_after_collection_ids:**
  An identifier specifying any dependencies or conditions for starting a collection after the completion of other collections, useful for sequencing and scheduling tasks.

- **17-vertical_scaling:**
  Refers to increasing or decreasing computational resources (such as CPU and RAM) for a system or collection to improve its performance. This process is automated, eliminating the need for users to specify exact resource requirements. In other words, the system determines the necessary resources for optimal performance.

- **18-scheduler:**
  The scheduler or scheduling algorithm used in the system, which affects how resources are prioritized and allocated for tasks and jobs.

- **19-start_time:**
  Indicates the start time of an event, task, or job during the tracking period, providing temporal information for analysis and monitoring.

- **20-end_time:**
  Indicates the end time of an event, task, or job, which allows for the calculation of duration and temporal relationships between events.

- **21-average_usage:**
  Shows the average resource consumption, such as CPU or memory, over a specific period, providing insights into resource usage trends.

- **22-maximum_usage:**
  Indicates the maximum resource consumption observed during a specific event, task, or job, reflecting peak load or resource demand.

- **23-random_sample_usage:**
  Provides data on the resource consumption of a random sample, offering a representative view of resource usage patterns.

- **24-assigned_memory:**
  Specifies the amount of memory allocated or assigned to a task, job, or resource sample, guiding resource allocation decisions.

- **25-page_cache_memory:**
  Indicates the memory used for page caching purposes, providing insights into system performance and efficiency.

- **26-cycles_per_instruction:**
  Shows the average number of processor cycles for each executed instruction, reflecting processor performance and efficiency.

- **27-memory_accesses_per_instruction:**
  Indicates the average number of memory accesses for each executed instruction, offering insights into memory usage patterns.

- **28-sample_rate:**
  Specifies the sampling rate or measurement frequency, which affects the accuracy and detail of resource consumption data.

- **29-cpu_usage_distribution:**
  Describes the distribution of CPU usage across different samples, tasks, or jobs, providing important insights into workload distribution and resource density. This distribution can be explained as follows:

  - **Workload Distribution:** This aspect shows how different tasks utilize CPU resources and which tasks may place the highest load on the CPU.
  - **Resource Density:** By examining the CPU usage distribution, one can identify which resources are more stressed and require optimization.
  - **Samples and Tasks:** Analyzing CPU usage among different samples or tasks provides detailed information about the performance of each part of the system.

Overall, analyzing CPU usage distribution can help identify performance issues and opportunities for improvement in resource management.

• 30-tail_cpu_usage_distribution:

Provides information about the tail distribution of CPU usage, indicating the frequency of extreme or anomalous usage patterns. This distribution can be described as follows:

- **Extreme Usage Patterns:** Identifies cases where CPU consumption is significantly higher than usual, which may be due to heavy workloads or inefficient programs.
- **Anomalous Usage Patterns:** Detects patterns that are unusual in terms of CPU consumption, such as sudden spikes or unexpectedly low usage.
- **Frequency of Patterns:** Examining the frequency of these extreme or anomalous patterns helps to better understand their impact on overall system performance.

Overall, analyzing the tail distribution of CPU usage can assist in identifying and managing performance issues and optimizing resource consumption.

- **31-event:**
  Refers to an occurrence or action within the system and provides a categorized description of system activities or changes.

- **32-failed:**
  Indicates whether a specific event, task, or job has failed, serving as a binary indicator for success or failure within the system.

- The purpose of the explanations provided so far was to give an overview of the dataset's columns so that we can identify the dataset's features and explain the reasons for the preprocessing steps we will undertake.

- In Data Pre-processing, we generally pursue several specific objectives to transform the original dataset into an optimized and usable one, ultimately enabling us to implement the model. These objectives include:

  1. Handling Missing Values and Nan
  2. Handling Noise & Outliers
  3. Removing Duplicate Data
  4. Normalization

  - In addition to these objectives, other processing steps will undoubtedly be necessary, which will be discussed further.

  - The steps we are referring to have all been applied to the dataset. Initially, we performed all four operations mentioned above as the first four stages, making changes to the dataset accordingly.

❖For 1-Handling Missing Values and Nan, which was the first operation we performed on the dataset, we found that a total of 5 columns contained Nan values:

resource_request : 744

vertical_scaling : 959

scheduler : 959

cycles_per_instruction : 124688

memory_accesses_per_instruction : 124688

❖Next, we will discuss separating the columns, and the **resource_request** column has this issue. The values in this column are in the following format:

{'cpus':0.2324, 'memory':0.42431}

- Therefore, we need to separate these values, which contain two types of data, by creating two new columns and placing the relevant information in each column. For this reason, the data must first be split, and then the Missing Values need to be handled.

- As a result, we split the **resource_request** column into two columns: **resource_request_cpus** and **resource_request_memory**, and removed the original **resource_request** column.

- Now, the Nan values in these two features need to be filled in a way that makes sense. Based on the analyses conducted, it's best to fill them with zeros. This is because in the original **resource_request** column, there were records where both **cpus** and memory values were zero (e.g., sample 1082 and others), and the number of Nan values is very small, so there's no need to use the mean or mode here.

- Now, let's examine the two features **vertical_scaling** and scheduler, both of which have the same number of missing data points.

- Upon observing these two features, we notice that the values within them are integers with very little variation. Since the **vertical_scaling** feature does not have a zero among its values, we decided to fill the Nan values with 1 in this feature, and with 0 in the scheduler feature. Again, it's important to note that the mean is not suitable here because the values are integers.

- Finally, we turn to **cycles_per_instruction** and **memory_accesses_per_instruction** , which contain a significant amount of missing data. In this case, filling these values with zero or a similar number would not be appropriate, so we need to use values similar to the mean to maintain balance within the data.

- In this way, the Missing Values in our dataset have been completely eliminated, and we have completed a very important step in data preprocessing.

❖ For **2 - Handling Noise & Outliers**, it should be noted that noise data in the dataset was reviewed and removed. However, some outlier data needs to remain in the dataset so that during visualization and model implementation, they can be distinguished from the clusters and observed.

❖ Additionally, after thorough analysis, we found that there is no duplicate data within the dataset.

❖ Moreover, as the final step in data preprocessing, normalization was also performed. We even applied normalization after the operations conducted later as part of **Feature Engineering** to ensure that all features are on a consistent and defined scale.

- Another issue that prompted us to make changes was that the dataset contained columns with multiple types of data, which forced us to separate them into two or more columns and then remove the original column to work with the newly created features.

- Earlier, we mentioned dividing the **resource_request** column into two features.

- Now, the three columns **average_usage**, **maximum_usage**, and **random_sample_usage** have a similar structure to the **resource_request** column. In these columns, the values are also formatted as follows:

{'cpus':0.2324, 'memory':0.42431}

- Therefore, we are forced to split each of these columns into two columns. As a result of the mentioned analysis, the above columns will be dropped from the dataset, and the following features will be generated:

average_usage_cpus

average_usage_memory

maximum_usage_cpus

maximum_usage_memory

random_sample_usage_cpus

random_sample_usage_memory

- The mentioned columns were not among the 5 columns containing Nan values, but the **random_sample_usage_memory** column was entirely Nan, and all values in this column were filled with zeros.
- Additionally, the newly created features were reviewed, and none of them contained Nan values. For now, we will keep this column in the dataset, and if necessary, we will remove it during model implementation.

- Now, we turn to the two columns **cpu_usage_distribution** and **tail_cpu_usage_distribution**. The values in these two columns also need to be separated. In the **cpu_usage_distribution** column, our data consists of a sequence of 11 numbers. These 11 numbers appear to be contained within a list and are separated from each other by a space without commas, as shown below:

[0.2 0.0 0.43 0.6 0.2 0.4 0.6 0.0 0.1 0.7 0.8]

- It is clear that the datatype of the values in these columns is string. Additionally, the values in the tail_cpu_usage_distribution column are identical in format to those in cpu_usage_distribution, except that the tail_cpu_usage_distribution column contains a list of 9 numbers instead of 11 numbers.

- Moreover, the Nans in these two columns are represented differently. According to our current analysis, neither of these columns contains missing data. Instead, Nan values are stored as [] for each record. Based on the analyses and preprocessing steps that will be mentioned later, it is found that both columns have Nans to the same extent. In other words, if a sample has a Nan value in **cpu_usage_distribution**, it will also have a Nan value in **tail_cpu_usage_distribution** for the same sample, and vice versa.

- Now, to separate these values, we plan to create new features and perform feature extraction. The individual numbers alone are not very useful, but the statistical information that can be extracted from them will be quite valuable.

For this purpose , we have added the following features to the dataset:

- From **cpu_usage_distribution** :

mean_cpu_usage -
max_cpu_usage -
min_cpu_usage -
variance_cpu_usage -
mode_cpu_usage -
median_cpu_usage -
max_min_diff_cpu_usage -

- From **tail_cpu_usage_distribution :**

mean_cpu_usage_tail_distribution -
max_cpu_usage_tail_distribution -
min_cpu_usage_tail_distribution -
variance_cpu_usage_tail_distribution -
mode_cpu_usage_tail_distribution -
median_cpu_usage_tail_distribution -
mean_cpu_usage_tail_distribution -
max_min_diff _cpu_usage_tail_distribution -

- All the features derived from the two columns have been created.

- We will definitely remove **cpu_usage_distribution** and **tail_cpu_usage_distribution**, and use the useful features we have generated. Another feature called covariance was also created. This feature was generated using the two columns by calculating the covariance between the first 9 numbers in the **cpu_usage_distribution** list and the numbers in the **tail_cpu_usage_distribution** list (which originally had 9 numbers). As we know, covariance measures the degree to which numbers vary together, so this is a valuable feature.

- The newly created features are self-explanatory, and their names indicate the statistical measures they contain. For example, **max_min_diff** in both columns refers to the range or spread of the numbers in each sample's list.

- Throughout all the changes described, the dataset is continuously updated and transformed to eventually reach an optimized and suitable final version. By the end, the dataset may have been updated in over 30 stages.

❖ The key point here is that our ultimate goal is to implement a multi-core fuzzy clustering model. For training such a model, converting non-numeric data to numeric data is essential because clustering models, especially this type, must be trained on numerical data.

❖ Therefore, we converted all categorical features in the dataset to numerical features. The dataset only contained one categorical feature, and we applied one-hot encoding to the "event" feature. As a result, the following features were obtained:

1-event_ENABLE

2-even_EVICT

3-event_FAIL

4-event_FINISH

5-event_KILL

6-event_LOST

7-event_QUEUE

8-event_schedule

9-event_UPDATE_PENDING

10-event_UPDATE_RUNNING

- In fact, what we have done now has resulted in one feature being removed and new features being created.

- We will continue to apply feature engineering and feature extraction to optimize the dataset. During these processes, the size of the dataset may vary, but it should be noted that the overall size will not change significantly and will remain around 300 MB.

- Another feature we created is **time_range**. This feature represents the time span of a task and is obtained by subtracting the **start_time** from the **end_time** of the same sample. Additionally, it can be concluded that the minimum value in this column is zero and that the feature does not contain any Nan values.

- The columns **start_after_collection_ids** and constraint were removed because they had a high volume of Nan values. It should be noted that missing data in these columns were also represented as [], effectively being empty lists.

- So far, all columns have been reviewed and features have been identified. Columns that were not analyzed were either named differently or had IDs that were not relevant for the training phase, so they remain untouched. These columns will not be included when the data is fed into the model for training, but there is no need to remove them from the dataset as they serve as identifiers for each sample.

- Additionally, all Nans in the newly added features have been checked and removed.