### **Introduction to Batch Systems**

#### **1. Definition of a Batch System:**

A **batch system** is a type of computing system where jobs (or tasks) are executed without interactive user intervention. Rather than being processed one at a time by a user, tasks are grouped together in "batches" and processed sequentially by the computer system.

In such systems, users typically submit jobs, which are then queued, and the system schedules and processes them as resources (like CPUs or memory) become available.

#### **2. Historical Context:**

Batch systems were prevalent in early computing environments, particularly in the 1950s to 1970s, when the computational resources were limited. Since users did not interact with the computer in real-time, multiple tasks could be processed during idle times or in between other jobs, increasing the efficiency of the system.

#### **3. How Batch Systems Work:**

The working principle of a batch system can be summarized in the following steps:

* **Job Submission**: Users submit their jobs in the form of scripts or command files. These jobs typically include the required computations and resources (e.g., CPU time, memory).
* **Job Queueing**: Once submitted, the jobs are placed in a queue by the batch system.
* **Job Scheduling**: The system scheduler decides which job to run next based on certain algorithms (e.g., First-Come-First-Served (FCFS), Shortest Job First (SJF), etc.).
* **Job Execution**: The batch system executes the jobs one after the other, based on available resources. These jobs do not require user intervention once submitted.
* **Completion and Output**: Once a job is executed, the system produces output (e.g., result files, logs) and informs the user of the job's completion, typically by writing results to a file.

#### **4. Types of Batch Systems:**

* **Offline Batch Systems**: In these systems, the jobs are submitted without the need for immediate processing. Jobs are queued, and when resources are available, the batch system processes them one by one, often without real-time feedback.
* **Online Batch Systems**: In contrast to offline systems, online batch systems handle batch jobs in real-time, where users can submit jobs dynamically, and the system handles the load as it comes in.

#### **5. Advantages of Batch Systems:**

* **Efficiency**: Batch systems can handle a large number of jobs in an efficient manner, reducing idle times of resources.
* **No User Interaction Needed**: Once jobs are submitted, no user intervention is required, allowing for unattended execution.
* **Cost-Effective**: It allows for the maximization of computing resources by scheduling tasks efficiently.
* **Ideal for Long-Running Jobs**: Batch systems are particularly useful for jobs that take a long time to complete and don’t require user interaction during execution (e.g., data processing, simulations).

#### **6. Disadvantages of Batch Systems:**

* **Lack of Real-Time Feedback**: Since jobs are processed without direct interaction, users don’t get immediate feedback during execution, which can be frustrating if there’s an issue.
* **Long Queues**: If many users submit jobs at once, the job queue can become long, and jobs might have to wait for an extended period before being processed.
* **No Dynamic Adjustments**: Some batch systems may not easily accommodate changes in job parameters during execution, as jobs are typically submitted with fixed parameters.

#### **7. Common Components of Batch Systems:**

* **Job Scheduler**: The component responsible for deciding which job to run next based on predefined policies or algorithms. Popular algorithms for scheduling include **First-Come-First-Served (FCFS)**, **Shortest Job First (SJF)**, and **Priority Scheduling**.
* **Queue**: A list or structure that holds the jobs waiting to be processed. Jobs are processed sequentially from the queue.
* **Job Monitor**: Monitors the progress of jobs, tracks their status, and records any output or error.
* **Job Executor**: The component that performs the actual computation for the jobs in the queue.

#### **8. Examples of Batch Systems:**

* **IBM JCL (Job Control Language)**: One of the most well-known batch processing systems used by IBM mainframe computers.
* **Slurm**: An open-source batch scheduling system used primarily in high-performance computing (HPC) environments to manage job queues on supercomputers and clusters.
* **Apache Hadoop**: While not a traditional batch system, Hadoop facilitates batch-style processing of big data by executing jobs in parallel across a distributed system.

#### **9. Batch vs Interactive Systems:**

* **Batch System**: Jobs are processed in groups with minimal user interaction. Users submit a job, and the system processes it without requiring real-time input.
* **Interactive System**: Users interact with the system in real-time, providing immediate input and receiving output during the process. Examples include desktop environments or command-line tools.

#### **10. Common Use Cases for Batch Systems:**

* **Data Processing**: Large-scale data transformation, cleaning, or aggregation jobs can be run in batches.
* **Simulation Jobs**: Many scientific and engineering computations (like simulations) are ideal for batch processing, as they typically take a long time to run and don’t need to interact with the user.
* **Data Analysis**: In fields like machine learning, batch systems can be used to run multiple experiments or analysis on large datasets without the need for real-time feedback.

#### **11. Modern Trends in Batch Systems:**

* **Cloud Computing**: Modern batch systems are often integrated with cloud environments where jobs can be distributed and executed in parallel across multiple virtual machines (VMs).
* **Hybrid Systems**: Some modern systems combine batch processing with interactive processing, allowing for a flexible system that can handle both types of jobs.

#### **12. Batch Processing in High-Performance Computing (HPC):**

In the context of HPC, batch systems are used to manage the efficient allocation of resources across multiple users and workloads. High-performance computing clusters often rely on batch systems to queue and manage jobs, ensuring that resources like CPUs, memory, and storage are allocated based on the needs of the job and the available system resources.

### **Conclusion:**

Batch systems are an essential part of computing environments where tasks can be grouped and processed without immediate interaction. Despite their lack of real-time feedback and possible delays in job processing due to queuing, batch systems offer an efficient and cost-effective way to manage jobs, especially in large-scale, long-running computational tasks. Their importance continues in modern systems like **Slurm** in high-performance computing, cloud infrastructure, and big data processing frameworks.

### **Resource Managers: Overview and Functionality**

#### **1. Definition of a Resource Manager:**

A **resource manager** is a system component or software that allocates, tracks, and manages the computing resources (like CPU, memory, disk space, etc.) in a shared computing environment. The primary role of a resource manager is to efficiently allocate resources to various tasks or jobs, ensuring that these resources are used optimally and equitably among users or processes.

Resource managers are especially critical in environments where many users or applications share a finite set of computing resources, such as in high-performance computing (HPC) clusters, data centers, and cloud environments.

#### **2. Key Responsibilities of a Resource Manager:**

* **Resource Allocation**: Ensures that jobs or tasks are assigned the necessary resources (e.g., CPU cores, memory, storage) based on availability and requirements.
* **Resource Monitoring**: Continuously tracks resource usage and availability across the system. It monitors whether the jobs are utilizing their allocated resources efficiently.
* **Scheduling Jobs**: The resource manager often works closely with the job scheduler to prioritize and queue tasks based on their resource needs, urgency, and job dependencies.
* **Job Management**: It is also responsible for managing the state of jobs (e.g., whether a job is running, waiting, or completed) and handling any issues like job failures or terminations.
* **Job Isolation and Security**: In multi-tenant environments, a resource manager may ensure that jobs do not interfere with one another and that resources are isolated to prevent conflicts.
* **Fairness and Load Balancing**: A resource manager strives to balance the load across different resources to ensure fair access for all users while preventing any single task or user from monopolizing resources.

#### **3. Types of Resource Managers:**

* **Centralized Resource Managers**: In this type, a single system manages and allocates all resources in a centralized manner. A master node or server might handle all requests and allocate resources from a central pool.
  + **Example**: SLURM, PBS, or SGE in HPC environments.
* **Distributed Resource Managers**: These resource managers are distributed across a network, with each node managing its own resources and contributing to a collective pool. This approach can offer greater flexibility and scalability.
  + **Example**: Kubernetes, Apache Mesos (in cloud environments).
* **Cloud-Based Resource Managers**: In cloud environments, resource managers dynamically allocate resources based on demand, scaling up or down as needed. These can manage virtual machines, storage, or containers in a cloud infrastructure.
  + **Example**: Amazon EC2, Google Cloud Compute Engine.

#### **4. How a Resource Manager Works:**

A resource manager typically interacts with the system’s scheduler and job execution components to manage resources. Below is a simplified process of how a resource manager typically functions:

1. **Resource Request**: A user or application submits a job requesting specific resources (e.g., "I need 4 CPU cores and 16 GB of memory").
2. **Job Queueing**: The resource manager places the job in a queue if the requested resources are not currently available.
3. **Resource Allocation**: When resources become available (e.g., when a task finishes and releases CPU cores or memory), the resource manager assigns the requested resources to the job from the queue.
4. **Job Execution**: Once the job is assigned resources, it is executed on the designated nodes or virtual machines.
5. **Monitoring**: Throughout job execution, the resource manager monitors resource usage, ensures fair allocation, and can adjust resource allocation if needed.
6. **Job Completion**: After job completion, the resource manager releases the resources and updates its resource state.
7. **Reporting**: It might also generate reports or logs of resource usage for billing, auditing, or analysis.

#### **5. Examples of Resource Managers:**

* **SLURM (Simple Linux Utility for Resource Management)**: SLURM is one of the most widely used open-source resource managers for managing large-scale compute clusters. It is designed to allocate resources (CPU, memory, etc.) to users, schedule jobs, and manage job queues. SLURM also supports fair share scheduling, load balancing, and priority management.
* **PBS (Portable Batch System)**: PBS is a job scheduler and resource manager used primarily in high-performance computing (HPC) environments. It ensures that jobs are allocated to available resources based on job priority and resource requirements. PBS has versions like OpenPBS and TORQUE, both widely used in academic and scientific communities.
* **Torque Resource Manager**: An open-source version of PBS, Torque provides job scheduling and resource management for clusters and supercomputers. Torque manages job submission, queuing, and scheduling to ensure resources are allocated effectively.
* **Apache Mesos**: Apache Mesos is a distributed resource manager for cloud computing environments. It is designed to scale to thousands of nodes and manage resources across a large cluster of machines. It can handle various workloads, including both batch and real-time tasks.
* **Kubernetes**: Kubernetes is a container orchestration platform that also serves as a resource manager for containerized applications. It manages resources for applications running in containers, automating the scaling, deployment, and resource allocation of those containers across nodes in a cluster.
* **Cloud-based Resource Managers (e.g., AWS EC2, Google Kubernetes Engine)**: In the cloud, resource managers dynamically allocate computing resources, such as virtual machines or containers, based on user demand. These systems often auto-scale, adjusting the available resources to meet the current workload.

#### **6. Resource Manager Algorithms:**

Resource managers often use algorithms to determine how resources should be allocated. Some common algorithms include:

* **First-Come, First-Served (FCFS)**: Jobs are processed in the order they are submitted.
* **Fair Share Scheduling**: Ensures resources are fairly distributed among users or jobs over time.
* **Priority-Based Scheduling**: Jobs are assigned priority levels, and higher-priority jobs are allocated resources first.
* **Backfilling**: This approach allows shorter jobs to "fill in" the gaps between longer jobs in the queue, improving resource utilization.
* **Proportional Share Scheduling**: Resources are divided based on proportional shares assigned to users or tasks.

#### **7. Challenges in Resource Management:**

* **Resource Contention**: Multiple users or jobs competing for the same resources can cause delays, requiring the resource manager to effectively handle conflicts.
* **Scalability**: As the number of nodes or resources increases, the resource manager needs to scale effectively to handle larger workloads.
* **Fairness**: In multi-user environments, ensuring that all users get fair access to resources without any individual monopolizing them is a challenge.
* **Resource Fragmentation**: Over time, resources may become fragmented, meaning small chunks of resources are not utilized efficiently. Resource managers need to manage this fragmentation to improve overall system efficiency.

#### **8. Modern Trends and Features in Resource Management:**

* **Dynamic Resource Allocation**: Modern resource managers, particularly in cloud and containerized environments, are increasingly able to allocate resources dynamically in response to changing workloads.
* **Integration with Cloud Platforms**: Many resource managers now integrate with cloud services, enabling hybrid environments where workloads can span both on-premise clusters and cloud infrastructure.
* **Containerization**: With the rise of containerized applications (e.g., using Docker), resource managers like Kubernetes have emerged to allocate resources to containers, providing a more flexible and isolated environment for applications.

### **Conclusion:**

A **resource manager** is an essential component in managing computing resources, especially in multi-user environments such as HPC clusters, data centers, and cloud platforms. It ensures that resources are allocated efficiently, jobs are scheduled properly, and the system remains fair and balanced. Whether in traditional batch environments, cloud computing, or modern containerized ecosystems, resource managers are crucial in maintaining system performance, maximizing resource utilization, and ensuring equitable access for users.

### **Schedulers: Overview and Functionality**

#### **1. Definition of a Scheduler:**

A **scheduler** is a software component or system responsible for determining which jobs or tasks should be executed at any given time, and how computing resources should be allocated to these tasks. In multi-tasking systems, the scheduler’s role is to prioritize and manage jobs in a way that maximizes system efficiency, ensures fairness, and meets the required performance criteria.

Schedulers are a critical part of both **resource management** and **job management** systems in environments like high-performance computing (HPC) clusters, cloud environments, and data centers. A scheduler works closely with a **resource manager** to allocate resources to jobs and determine when those jobs should run based on factors like job priority, resource availability, and scheduling policies.

#### **2. Types of Schedulers:**

Schedulers can be broadly classified into two types:

* **Job Scheduler**: Focuses on scheduling batch jobs that need to be executed on a system. A job scheduler determines the order in which batch jobs should be executed and allocates the necessary resources.
* **Process/Task Scheduler**: A low-level scheduler responsible for scheduling individual processes or threads within an operating system. This is more about allocating CPU time to processes running on the system.

In the context of **HPC systems** and **distributed computing**, we are primarily concerned with **job schedulers**.

#### **3. How Job Schedulers Work:**

Job schedulers typically work in the following sequence:

1. **Job Submission**: A user submits a job, often specifying required resources such as the number of CPU cores, memory, and runtime.
2. **Job Queuing**: The job is placed in a queue, waiting for available resources to be allocated. The job remains in the queue until resources become available.
3. **Job Scheduling**: The scheduler selects jobs from the queue based on specific criteria and allocates resources to those jobs. This process involves prioritization, load balancing, and resource management.
4. **Job Execution**: Once the job has been assigned resources, it starts execution. The scheduler ensures that the resources are available and the job runs on the correct node or system.
5. **Job Monitoring**: During execution, the scheduler may monitor the job's progress, adjusting resource allocation if necessary (e.g., to avoid overloading the system).
6. **Job Completion**: Once the job finishes, the scheduler releases the resources and updates the job status. The job's output is made available to the user, and the scheduler prepares for the next job in the queue.

#### **4. Common Job Scheduler Algorithms:**

Schedulers often use various algorithms to determine the order in which jobs should be executed. Some common job scheduling algorithms include:

* **First-Come, First-Served (FCFS)**: Jobs are executed in the order they arrive in the queue. This is a simple scheduling algorithm but may not be optimal in all situations.
* **Shortest Job First (SJF)**: This algorithm prioritizes jobs that require the least amount of computational time. It helps minimize average waiting time but may lead to starvation of longer jobs.
* **Priority Scheduling**: Each job is assigned a priority, and jobs with higher priority are executed first. This algorithm allows for prioritizing critical tasks but may suffer from starvation if low-priority jobs are constantly preempted.
* **Round Robin (RR)**: Jobs are allocated fixed time slots (quantum), and each job is given a turn to execute. This ensures fairness and responsiveness, but it may not be ideal for long-running jobs.
* **Multilevel Queue Scheduling**: Jobs are classified into different priority queues, and each queue uses a different scheduling algorithm (e.g., FCFS for low-priority jobs, SJF for high-priority jobs). This allows for a flexible scheduling approach based on job characteristics.
* **Fair Share Scheduling**: This approach ensures that resources are allocated fairly among users or groups based on their share or allocation. It can be used in environments where multiple users or projects are competing for resources.

#### **5. Types of Job Schedulers:**

* **Simple Batch Schedulers**: These are basic schedulers that accept job submissions and allocate resources to jobs in a straightforward manner, often using simple algorithms like FCFS or Round Robin. Simple schedulers are common in small-scale or non-HPC environments.
* **Advanced Job Schedulers**: These schedulers are much more complex and can handle large-scale clusters and complex job dependencies. They often support sophisticated scheduling policies such as fair-share scheduling, backfilling, and priorities based on resource usage or job characteristics.  
  + **Examples**:
    - **SLURM**: A highly scalable and flexible job scheduler used in HPC environments. SLURM allows for fine-grained resource allocation, job priorities, and support for parallel jobs.
    - **PBS (Portable Batch System)**: A job scheduler that works with distributed computing environments. PBS supports resource allocation, job queuing, and scheduling based on different policies.
    - **Grid Engine (SGE)**: A job scheduling system that is often used for managing distributed computing resources in a cluster. It provides features like resource reservations and job dependencies.
* **Cloud-based Job Schedulers**: In cloud environments, job schedulers are often integrated with cloud services to allocate virtual machines or containers for jobs. These schedulers can scale dynamically based on the demand for computational resources.  
  + **Examples**:
    - **Kubernetes**: A container orchestration system that includes a built-in scheduler for allocating containers to available nodes in a cluster.
    - **Amazon Batch**: A fully managed batch processing service that schedules and runs jobs on AWS, dynamically provisioning computing resources as needed.

#### **6. Key Features of Advanced Job Schedulers:**

* **Job Prioritization**: Allows jobs to be prioritized based on various factors such as urgency, job type, or user importance.
* **Fairness and Fair Share Scheduling**: Ensures that multiple users or tasks have equitable access to system resources, preventing any single job or user from monopolizing resources.
* **Backfilling**: A technique used to improve resource utilization by allowing small, shorter jobs to “fit in” the gaps between long-running jobs in the queue, thus reducing waiting time.
* **Job Dependencies**: Some jobs may depend on the completion of other jobs (e.g., in a workflow). Advanced schedulers can manage these dependencies and ensure that jobs are executed in the correct order.
* **Resource Constraints**: Jobs can specify resource requirements (e.g., number of CPUs, amount of memory), and the scheduler ensures that these requirements are met before execution.

#### **7. Challenges in Job Scheduling:**

* **Job Starvation**: In some algorithms like priority scheduling, lower-priority jobs might never get executed if higher-priority jobs keep arriving. Schedulers need to implement techniques to prevent starvation.
* **Load Balancing**: A key challenge in large clusters is ensuring that resources are evenly distributed across nodes. Uneven load balancing can lead to some nodes being overutilized while others remain idle.
* **Scalability**: As the number of jobs or nodes increases, the scheduling system must scale accordingly. Large-scale systems require sophisticated scheduling algorithms to ensure optimal performance.
* **Resource Fragmentation**: Over time, resources can become fragmented (e.g., small unused blocks of memory), which reduces the efficiency of the system. The scheduler must manage resources effectively to avoid fragmentation.
* **Job Dependencies**: Managing complex job dependencies (e.g., a workflow of multiple jobs that must run in a specific order) can be challenging, especially when jobs fail or are delayed.

#### **8. Modern Trends in Job Scheduling:**

* **Hybrid Scheduling**: Modern schedulers often use hybrid approaches, combining different scheduling algorithms depending on job type, priority, and system load.
* **Elastic Scheduling**: In cloud environments, schedulers dynamically allocate resources based on fluctuating demand, enabling autoscaling of virtual machines or containers to meet workload demands.
* **Scheduling for Containers**: With the rise of containerized applications, scheduling tools like **Kubernetes** and **Docker Swarm** allow for the efficient allocation and management of containers across a cluster of nodes.

### **Conclusion:**

A **job scheduler** is an essential part of resource management in computing systems, responsible for deciding when and where jobs should run based on factors such as priority, resource availability, and job characteristics. By implementing various scheduling algorithms, a scheduler ensures fair allocation of resources, maximizes system performance, and minimizes job delays. From simple batch schedulers to sophisticated systems in cloud environments and HPC, schedulers are critical for managing computational resources efficiently and ensuring that jobs execute optimally.

### **Job Submission, Management, and Writing Batch Scripts**

In computing environments, especially in **high-performance computing (HPC)** clusters and **cloud computing** environments, job submission, management, and writing batch scripts are critical aspects of effectively utilizing available resources. These concepts are particularly relevant in environments where **batch systems** like **SLURM** or **PBS** are used. Here's an in-depth look at these topics.

### **1. Job Submission**

#### **What is Job Submission?**

Job submission refers to the process where users submit tasks or programs (known as "jobs") to a job scheduling system (like **SLURM**, **PBS**, or **SGE**) for execution. When you submit a job, you tell the system what resources your job requires, and the system places it in a queue to be scheduled and executed once the required resources are available.

#### **Job Submission Process:**

* **Define the Job Requirements**: This includes specifying resource needs (e.g., number of CPUs, amount of memory, disk space), time limits, and other requirements for the job.
* **Job Submission Command**: Most batch systems provide a command to submit a job. For example, in **SLURM**, the command is sbatch, and in **PBS**, it is qsub.
* **Job Submission Syntax**: The syntax usually includes parameters for resource requests (CPU, memory), time limits, and the job script.
* **Job Queuing**: Once the job is submitted, it is placed in a queue until the system scheduler assigns resources and executes it.
* **Job Monitoring**: After submission, users can monitor the job's progress using commands like squeue (SLURM), qstat (PBS), or scontrol (SLURM).
* **Job Completion and Output**: Once the job completes, output files (such as result files or logs) are generated. These can be retrieved by the user, and the job is removed from the queue.

#### **Example (Using SLURM):**

sbatch --nodes=2 --ntasks=4 --time=02:00:00 my\_job\_script.sh

This command submits a job that requests:

* 2 nodes
* 4 tasks (processes)
* 2 hours for execution
* The script to run is my\_job\_script.sh.

### **2. Job Management**

#### **What is Job Management?**

Job management refers to overseeing the lifecycle of jobs within a scheduling system. It involves monitoring, controlling, and modifying jobs while they are in the queue or executing.

#### **Key Aspects of Job Management:**

* **Monitoring Job Status**: You can check whether your job is running, queued, or has completed. Monitoring tools like squeue (SLURM), qstat (PBS), or showq (SGE) are commonly used.
* **Controlling Jobs**: You may need to pause, resume, cancel, or hold jobs. These controls allow you to manage jobs during execution:
  + **Hold a Job**: Prevents a job from running until the hold is released. For example, in SLURM, scontrol hold <job\_id>.
  + **Cancel a Job**: Terminates a job before it completes. For example, in SLURM, scancel <job\_id>.
* **Job Dependencies**: Sometimes, jobs need to depend on other jobs before they can start. This is particularly useful in workflows where the output of one job is required for the next. In SLURM, you can specify job dependencies using --dependency=afterok:<job\_id>.
* **Job Priorities**: Many job schedulers assign priority to jobs based on factors like submission time, job size, or user quota. Administrators may configure priority-based scheduling to manage resource distribution fairly among users.

#### **Example:**

**Checking the Status of a Job** (SLURM):  
 squeue -u username

* **Cancel a Running Job** (SLURM):  
   scancel <job\_id>

### 

### 

### **3. Writing Batch Scripts**

#### **What is a Batch Script?**

A **batch script** is a text file that contains a series of commands that the job scheduler will execute when the job is started. The script defines the job's execution environment, including resource requests, environment variables, job parameters, and the commands that should be run.

Batch scripts are essential for automating tasks and submitting jobs in a consistent way, particularly in large-scale environments like HPC clusters.

#### **Key Components of a Batch Script:**

* **Resource Requests**: The script should include requests for the resources required by the job, such as CPU cores, memory, wall time, and nodes.
* **Job Commands**: The commands to be executed as part of the job, including running the program or script that is the primary task.
* **Output Directives**: The locations where standard output (stdout) and standard error (stderr) should be written.
* **Job Name and Time Limits**: Setting a job name, execution time limits, and other job attributes.

#### **Structure of a Batch Script:**

Batch scripts generally have a specific structure. The first part typically contains directives for the job scheduler, followed by the actual commands for the job. These directives depend on the scheduler used, but common directives include requests for nodes, CPUs, memory, time, and job output.

### **Example 1: SLURM Batch Script (my\_job\_script.sh)**

#!/bin/bash

#SBATCH --job-name=my\_job # Job name

#SBATCH --output=output.txt # Standard output file

#SBATCH --error=error.txt # Standard error file

#SBATCH --ntasks=1 # Number of tasks (processes)

#SBATCH --cpus-per-task=4 # CPU cores per task

#SBATCH --mem=8GB # Memory per node

#SBATCH --time=02:00:00 # Maximum runtime

#SBATCH --partition=standard # Partition (queue) to submit to

# Job commands below this line

echo "Job started at $(date)"

module load python/3.8.5 # Example: load a required module

python my\_script.py # Execute the script

echo "Job finished at $(date)"

In this script:

* #!/bin/bash: This shebang tells the system to use the Bash shell to execute the script.
* #SBATCH directives specify various job parameters (job name, resources, etc.).
* The job itself runs a Python script (my\_script.py).

#### **Example 2: PBS Batch Script (my\_job\_script.pbs)**

#!/bin/bash

#PBS -N my\_job # Job name

#PBS -o output.txt # Standard output file

#PBS -e error.txt # Standard error file

#PBS -l nodes=1:ppn=4 # 1 node with 4 processors per node

#PBS -l mem=8GB # Memory per node

#PBS -l walltime=02:00:00 # Maximum runtime

# Job commands below this line

cd $PBS\_O\_WORKDIR # Change to the directory where the job was submitted

echo "Job started at $(date)"

module load python/3.8.5 # Example: load a required module

python my\_script.py # Execute the script

echo "Job finished at $(date)"

In this script:

* #PBS -N my\_job: Specifies the job name.
* #PBS -l nodes=1:ppn=4: Requests 1 node with 4 processors per node.
* cd $PBS\_O\_WORKDIR: Changes to the directory from which the job was submitted.

### **4. Job Submission, Management, and Writing Batch Scripts Best Practices**

#### **Best Practices for Job Submission:**

* **Accurate Resource Requests**: Always request only the resources that are necessary for your job. Over-requesting resources can waste system resources, while under-requesting can lead to job failure.
* **Efficient Job Queuing**: Be mindful of queue wait times. Large jobs may take longer to be scheduled, so consider breaking them into smaller tasks when possible.
* **Error Handling**: Include error checking and logging in your batch script to identify and troubleshoot potential issues during job execution.
* **Job Dependencies**: Use job dependencies to ensure that jobs execute in the correct order, especially in workflows where the output of one job is needed for another.
* **Clean Output**: Ensure your script generates and stores output in an organized manner. Clean and structured output can be important for debugging and tracking job performance.

#### **Best Practices for Writing Batch Scripts:**

* **Modularize Your Scripts**: Keep scripts modular by separating different parts of the task into functions or separate scripts.
* **Use Environment Modules**: Many HPC systems use environment modules to manage software. Use them to ensure that the required software and libraries are loaded correctly before executing your job.
* **Time Management**: Always set a reasonable maximum wall time (walltime) to prevent jobs from running indefinitely. This ensures that resources are available for other users.
* **Check for Existing Jobs**: Before submitting a job, check whether similar jobs are already running or queued, especially in shared environments.

### **Conclusion:**

**Job submission, management, and writing batch scripts** are essential skills for utilizing computing resources in environments like HPC clusters or cloud platforms. Batch scripts allow users to automate job submission, specify resource requirements, and execute jobs efficiently. Effective job management ensures that users can monitor, control, and optimize job execution. By adhering to best practices and understanding the job scheduling system, users can maximize system utilization and reduce errors in a computational environment.

### **SLURM Installation and Configuration: A Comprehensive Guide**

**SLURM (Simple Linux Utility for Resource Management)** is an open-source, highly scalable resource manager used in many high-performance computing (HPC) environments. It handles job scheduling, resource allocation, and job management in a cluster. Setting up SLURM involves installing the software, configuring it to work with your hardware, and then integrating it with job scripts to manage computing resources efficiently.

### **1. Overview of SLURM Components**

SLURM is a modular system, which means it has various components that work together:

* **SLURM Controller (slurmctld)**: Manages the overall operation of the cluster, including job scheduling, allocation, and communication between different nodes.
* **SLURM Daemon (slurmd)**: Runs on each compute node. It handles the job execution and resource management tasks on each individual node.
* **SLURM Database Daemon (slurmdbd)**: Optional component used for job accounting and logging. It interfaces with a database to store historical job data.
* **SLURM User Commands**: Utilities that users and administrators use to interact with the SLURM system, such as sbatch (for submitting jobs), squeue (for job status), scontrol (for managing jobs), and sacct (for accounting).

### **2. Prerequisites for Installing SLURM**

Before installing SLURM, ensure that the following prerequisites are met:

* **Operating System**: A supported Linux distribution (CentOS, Ubuntu, etc.).
* **Dependencies**: Make sure the necessary dependencies are installed on your system. These may include:
  + gcc (GNU Compiler Collection)
  + make (for building the software)
  + munge (for authentication)
  + perl (for scripting)
  + MySQL or MariaDB (if you're using SLURM's database for accounting)
* **Network Configuration**: Ensure the nodes in the cluster can communicate with each other and that each node has a unique hostname.

### **4. SLURM Configuration (slurm.conf)**

SLURM configuration is handled through the slurm.conf file, which defines the hardware resources, job queues, and other SLURM-specific settings. A typical slurm.conf includes information about the nodes in the cluster, partitions (queues), and other parameters.

Here’s an example of a simple slurm.conf file:

# slurm.conf file for a simple cluster

# General settings

ControlMachine=slurmctl.example.com # The controller node

AuthType=auth/munge # Authentication method (munge)

SlurmdPort=7003 # SLURM daemon port

SlurmctldPort=7002 # SLURM controller port

SlurmdUser=root # User for slurmd processes

# Nodes

NodeName=node[01-10] CPUs=8 State=UNKNOWN # Define 10 nodes with 8 CPUs each

PartitionName=debug Nodes=node[01-10] Default=YES MaxTime=00:30:00 State=UP # Partition definition

# Optional: Accounting settings (requires slurmdbd)

AccountingStorageType=accounting\_storage/slurmdbd

AccountingStorageHost=slurmdbd.example.com

#### **Key Parameters in slurm.conf:**

* **ControlMachine**: Specifies the name of the SLURM controller (master node).
* **NodeName**: Defines the nodes in the cluster and their resource attributes (like CPU count).
* **PartitionName**: Defines job queues (partitions) with specified nodes, maximum time limits, and state.
* **SlurmctldPort / SlurmdPort**: Port numbers for communication between the controller and the compute nodes.
* **AccountingStorage**: Defines the accounting backend for tracking resource usage.

**Conclusion**

Installing and configuring SLURM involves several key steps: preparing the environment, installing the necessary components (including Munge for authentication), configuring SLURM's core components (`slurm.conf`), and enabling job accounting with `slurmdbd` (optional). Once installed, SLURM provides a powerful, flexible resource management system for job scheduling, resource allocation, and accounting in HPC environments.

### **Submitting and Managing Jobs in SLURM**

In SLURM (Simple Linux Utility for Resource Management), **job submission** and **job management** are crucial activities for users interacting with the system to utilize computational resources efficiently. In this section, we'll cover how to **submit jobs**, monitor their status, and manage them (cancel, suspend, or requeue).

### **1. Job Submission in SLURM**

To submit a job to SLURM, you typically write a **batch script** that contains job parameters, resources required, and the commands to execute. You then submit this script using the sbatch command.

#### **Job Submission Steps:**

**Create a Batch Script**: A batch script is a text file containing job directives and commands to run. These scripts typically start with a few **SLURM directives** that specify job resources and other configurations, followed by the actual commands that need to be executed.  
  
 Here's a basic example of a batch script (my\_job\_script.sh):  
  
 #!/bin/bash

#SBATCH --job-name=my\_job # Job name

#SBATCH --output=output.txt # Standard output file

#SBATCH --error=error.txt # Standard error file

#SBATCH --ntasks=1 # Number of tasks (processes)

#SBATCH --cpus-per-task=4 # CPU cores per task

#SBATCH --mem=8GB # Memory per node

#SBATCH --time=02:00:00 # Maximum wall time (HH:MM:SS)

#SBATCH --partition=standard # Partition to submit the job to

# Job commands below this line

echo "Starting job on $(date)"

python my\_script.py

echo "Job finished on $(date)"

1. In the above script:  
   * #SBATCH directives specify the job name, output files, resources (CPUs, memory), and time limit.
   * The job will run the Python script my\_script.py.

**Submit the Job**: Once you have the batch script ready, you submit it using the sbatch command.  
  
 sbatch my\_job\_script.sh

The sbatch command submits the job to SLURM. It returns a job ID, which you can use to track the job's status later.  
  
 Example:  
  
 Submitted batch job 12345

1. This means your job has been successfully submitted with the job ID 12345.

### **2. Job Status and Monitoring**

Once a job is submitted, SLURM places it in a job queue, and you can check its status using various SLURM commands.

#### **Common SLURM Job Monitoring Commands:**

**squeue**: View the status of jobs in the SLURM queue.  
  
 squeue

1. This command shows the following details:  
   * Job ID
   * Job name
   * Partition
   * Node(s) allocated
   * Job state (e.g., pending, running)
   * Time elapsed
   * Job owner

Example output:  
  
 JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)

12345 standard my\_job user1 R 01:20 2 node01,node02

1. **scontrol**: For detailed job status and management, scontrol provides more control.  
     
    To view detailed information about a job:  
     
    scontrol show job <job\_id>

Example:  
  
 scontrol show job 12345

1. **sacct**: This command provides historical job accounting information, such as job status, resource usage, and exit codes, after the job has finished.  
     
    To display the status of completed jobs:  
     
    sacct -j <job\_id>

Example:  
  
 sacct -j 12345

Example output:  
  
 JobID JobName Part AllocCPUs State ExitCode

12345 my\_job standard 8 COMPLETED 0:0

1. **sstat**: Display status information for running jobs.  
     
    Example:  
     
    sstat -j 12345.batch

This will give you detailed resource usage (e.g., CPU, memory) for a running job.

### **3. Job Management in SLURM**

Once a job is submitted, you may need to manage it, such as canceling, suspending, or requeuing a job. Here are some of the most common job management tasks in SLURM.

#### **Common SLURM Job Management Commands:**

**Cancel a Job (scancel)**: If you need to cancel a job that is running or pending, use the scancel command, followed by the job ID.  
  
 Example:  
  
 scancel 12345

1. This command will terminate the job with ID 12345. If the job is running, it will be immediately stopped.
2. **Suspend/Resume a Job (scontrol)**: If you want to suspend a running job (for example, to free up resources temporarily), you can suspend it using scontrol and later resume it.

Suspend a job:  
  
 scontrol suspend <job\_id>

Resume a suspended job:  
  
 scontrol resume <job\_id>

**Requeue a Job (scontrol)**: If a job has failed or been canceled, you can requeue it to be rescheduled for execution again.  
  
 scontrol requeue <job\_id>

**Hold and Release Jobs (scontrol)**: If you want to hold a job (to prevent it from running), you can set a hold, and later release it when you're ready.

Set a hold on a job:  
  
 scontrol hold <job\_id>

Release a hold on a job:  
  
 scontrol release <job\_id>

**View Job Logs**: For debugging purposes, it’s important to check the standard output and error logs generated by the job. These logs are specified in the batch script using #SBATCH --output and #SBATCH --error.  
  
 Example (if you specified #SBATCH --output=output.txt):  
  
 cat output.txt

**Job Dependencies**: If you have multiple jobs that need to run in a specific order, you can set dependencies. For example, a job might need to wait for another job to complete before starting.

To submit a job that depends on the completion of another job:  
  
 sbatch --dependency=afterok:<job\_id> my\_job\_script.sh

* + In this example, my\_job\_script.sh will only be submitted after the job with ID <job\_id> completes successfully.

### **4. Job Output and Debugging**

While managing jobs, it’s essential to handle the job output properly. SLURM allows you to specify **output** and **error** files where the stdout and stderr of the job are logged.

**Standard Output and Error**: You can define the output and error files using #SBATCH directives in your batch script:  
  
 #SBATCH --output=output.txt # Standard output file

#SBATCH --error=error.txt # Standard error file

1. This will log:  
   * **Standard output**: Regular output of the job (e.g., from echo or script execution).
   * **Standard error**: Error messages, such as runtime errors in the program.
2. **Job Exit Codes**: When a job completes, it returns an exit code. By checking the exit code, you can determine if the job was successful:  
   * **Exit Code 0**: Job completed successfully.
   * **Non-zero Exit Code**: The job encountered an error. For debugging, check the error file.
3. You can view the exit code using sacct or in the job output logs.

### **Conclusion**

**Submitting jobs** in SLURM is a simple process using the sbatch command and batch scripts. After submission, job management allows you to monitor and control jobs using tools like squeue, scontrol, and sacct. SLURM offers flexibility in handling job dependencies, suspending or resuming jobs, canceling jobs, and managing resources efficiently. By properly configuring job scripts and utilizing SLURM’s powerful job management capabilities, users can optimize job execution in high-performance computing environments.

### **Managing Computing Nodes and Configuring Server Scheduling Policies in SLURM**

In SLURM (Simple Linux Utility for Resource Management), effective **node management** and **server scheduling policies** are crucial for ensuring efficient resource allocation, job scheduling, and cluster management. This section will cover how to manage the compute nodes in a SLURM-managed cluster and configure scheduling policies to optimize job execution.

### **1. Managing Computing Nodes in SLURM**

Computing nodes in SLURM are managed through the slurmd daemon, which runs on each compute node and communicates with the SLURM controller (slurmctld). SLURM provides a set of commands to view, configure, and manage these nodes.

#### **a. Node Configuration in SLURM**

Node configuration is handled through the **slurm.conf** file, where administrators define the properties of each node. The configuration allows you to specify resources (CPUs, memory, etc.), node availability, and other relevant attributes.

Here is an example of node configuration in slurm.conf:

# Define nodes

NodeName=node[01-10] CPUs=8 State=UNKNOWN

NodeName=node11 CPUs=16 State=UNKNOWN

# Define partition (group of nodes)

PartitionName=debug Nodes=node[01-10] Default=YES MaxTime=01:00:00 State=UP

In this example:

* **NodeName=node[01-10]** defines a range of 10 nodes, node01 to node10, each with 8 CPUs.
* **CPUs=8** specifies the number of CPU cores available on each node.
* **State=UNKNOWN** means the node is not yet in an active state (can be updated to UP, DOWN, or other states).
* **PartitionName=debug** defines a partition for jobs to run, in this case, using the nodes node01 to node10.

#### **b. Check Node Status**

You can use the scontrol and sinfo commands to view the status of nodes:

**scontrol show node**: Displays detailed information about a specific node.  
  
 scontrol show node node01

1. This will show the current state, available resources, and other information for the node node01.

**sinfo**: Displays summary information about nodes and partitions.  
  
 sinfo

Example output:  
  
 PARTITION AVAIL TIMELIMIT NODES STATE NODELIST

debug up 01:00:00 10 idle node[01-10]

1. In the above example, 10 nodes are available in the debug partition, and they are idle.

**scontrol show node**: Displays detailed status for a specific node:  
  
 scontrol show node node01

1. This command shows detailed node information, including CPU allocation, memory usage, and node state (e.g., UP, DOWN).

#### **c. Change Node State**

You can manually change the state of a node to enable or disable jobs from running on it.

**Set a node to DOWN**: Prevents the node from being used for job scheduling.  
  
 scontrol update NodeName=node01 State=DOWN

* **Set a node to UP**: Marks the node as available for job scheduling.  
    
   scontrol update NodeName=node01 State=UP
* **Set a node to DRAIN**: Marks the node as temporarily unavailable, usually for maintenance.  
    
   scontrol update NodeName=node01 State=DRAIN
* **Set a node to IDLE**: Marks the node as idle, available for new jobs.  
    
   scontrol update NodeName=node01 State=IDLE

#### **d. Manage Node Resources**

1. **Disable or Enable CPUs**: SLURM allows you to disable or enable specific CPUs on a node if you want to reserve resources for specific tasks or perform maintenance.

Disable a CPU:  
 scontrol update NodeName=node01 CPUs=4

Enable all CPUs:  
 scontrol update NodeName=node01 CPUs=8

1. **Assign Specific Resources**: You can also assign specific resources like memory or GPUs to individual jobs, which can be defined in the job script or node configuration.

### **2. Configuring Server Scheduling Policies**

SLURM allows administrators to configure the **scheduling policies** that determine how jobs are queued and executed on the cluster. These policies can control job prioritization, fairness, resource allocation, and scheduling behavior.

#### **a. Scheduling Algorithms**

SLURM supports several scheduling algorithms, and the most common one is the **Fair Share Scheduler**. However, you can configure other algorithms for specific requirements.

The **Fair Share Scheduler (FSS)** dynamically adjusts job priorities based on historical resource usage. This ensures fair resource distribution across all users.

To configure the scheduling algorithm, you would edit the slurm.conf file to include the desired algorithm.

Example configuration:

# Scheduler settings

SchedulerType=sched/backfill # Use backfill scheduling algorithm

SchedulerPort=7003

* **Backfill Scheduling**: Jobs are scheduled in a way that maximizes resource utilization while respecting job priorities. The backfill scheduler will start smaller jobs in available gaps without delaying high-priority jobs.
* **Priority-based Scheduling**: SLURM also supports priority-based job scheduling, where jobs are executed based on a priority score (calculated from factors like job age, user group, job size, etc.).

#### **b. Partition Configuration**

A **partition** in SLURM represents a group of nodes that share similar resources. You can configure partitions to set limits on the type of jobs they can handle, the number of nodes to be used, and job priority.

In slurm.conf, you can define partitions with specific parameters, such as:

* **MaxTime**: Maximum allowed job runtime.
* **Default**: The default partition for job submission.
* **State**: The state of the partition (e.g., UP, DOWN).
* **Nodes**: The nodes included in the partition.

Example of a partition definition:

PartitionName=debug Nodes=node[01-10] Default=YES MaxTime=01:00:00 State=UP

This partition configuration defines a debug partition with 10 nodes, where jobs are allowed to run for up to 1 hour.

#### **c. Fair Share Scheduling**

Fair Share Scheduling is a key feature for balancing resource usage across users and projects. SLURM assigns a priority score to each user, and the priority changes over time based on job history.

You can configure fair share scheduling by enabling it in the slurm.conf file.

Example:

# Enable Fair Share Scheduling

PriorityType=priority/fairshare

PriorityDecayHalfLife=7-00:00:00 # Time to decay job priorities by half

PriorityWeightJobAge=1000 # Weighting for job age in priority calculation

PriorityWeightFairshare=1000 # Weighting for fair share in priority calculation

This configuration ensures that users with a lower resource usage history are given higher priority for job execution.

#### **d. Preemption and Job Prioritization**

SLURM allows administrators to configure preemption policies to prioritize certain jobs over others. Preemption means that higher-priority jobs can preempt (interrupt) lower-priority jobs if resources are needed.

You can define job preemption settings in the slurm.conf file. For example, you can set a **priority weight** to prioritize jobs based on specific criteria, such as job size, user, or resource usage.

Example configuration:

# Preempt jobs with lower priority

PreemptType=preempt/partition\_prio

PreemptMode=SUSPEND,REQUEUE

This setting enables preemption, where jobs with lower priority can be suspended and requeued in favor of higher-priority jobs.

### **3. Advanced Scheduling Policies**

SLURM provides a variety of advanced features to fine-tune scheduling, including:

**Job Arrays**: SLURM allows you to submit a series of similar jobs as a job array. The array jobs are scheduled in parallel based on available resources.  
  
 Example submission of a job array:  
  
 sbatch --array=1-10 my\_job\_script.sh

1. **Resource Constraints**: You can enforce resource limits, such as limiting the amount of memory, CPU, or GPUs that can be allocated to a job.  
     
    Example resource constraints in a job script:  
     
    #SBATCH --mem=4GB

#SBATCH --gres=gpu:2

1. **Quality of Service (QoS)**: SLURM allows you to define **Quality of Service (QoS)** levels to prioritize certain jobs or users.  
     
    Example configuration:  
     
    # Define QoS levels

QoSType=standard

QOS=high\_priority

1. **Job Time Limits**: You can specify time limits for jobs, ensuring that long-running jobs do not hold up resources for too long.  
     
    Example time limit in slurm.conf:  
     
    # Define job time limits for a partition

MaxTime=04:00:00 # Maximum allowed time for jobs in this partition

### **Conclusion**

Managing **computing nodes** and configuring **server scheduling policies** are essential components of SLURM's resource management capabilities. By configuring the slurm.conf file correctly and using SLURM's job scheduling policies, administrators can ensure efficient resource allocation, job execution, and fair usage among users. Proper node management allows administrators to monitor, maintain, and control node states and resources. Configuring scheduling policies enables control over job prioritization, preemption, resource allocation, and fairness, ultimately improving the overall efficiency and utilization of the cluster.

### **Understanding Different Scheduler Algorithms in SLURM**

SLURM (Simple Linux Utility for Resource Management) offers a variety of **scheduler algorithms** that determine how jobs are scheduled and prioritized within a cluster. These algorithms are crucial for optimizing resource utilization, managing workloads efficiently, and ensuring fairness among users and jobs.

In SLURM, the **scheduler** is responsible for deciding which jobs run on which nodes at any given time, based on available resources, job priority, and policies set by administrators. Let’s dive into the different types of **scheduler algorithms** supported by SLURM.

### **1. Backfill Scheduling Algorithm**

The **Backfill Scheduler** is the most commonly used scheduling algorithm in SLURM, particularly for large clusters where jobs of varying sizes and runtimes are submitted.

#### **How Backfill Works:**

* The backfill scheduler attempts to **maximize resource utilization** by filling available gaps in the schedule with smaller jobs without delaying larger, higher-priority jobs.
* It considers **job priorities** and **resources** to backfill gaps between jobs that have already been scheduled. Jobs with lower resource requirements may start earlier than their priority would normally allow if there is a time gap before a larger job starts.

#### **Example Workflow:**

1. **High-priority job** A is submitted and scheduled, but it can’t run until a large resource block becomes available.
2. In the meantime, **lower-priority job** B with smaller resource requirements can "backfill" into the available gap between the high-priority job and the larger block of resources.
3. This ensures that resources are **fully utilized** and the cluster does not remain idle.

#### **Backfill Configuration:**

The backfill scheduler is typically enabled by default in SLURM, and it works in combination with **job priorities** and **job age** to determine whether a job can backfill into the schedule. It’s configured in the slurm.conf file under the SchedulerType parameter:

SchedulerType=sched/backfill

### 

### **2. Priority Scheduling Algorithm**

The **Priority Scheduler** is used to assign a priority score to each job and then schedule jobs based on their priority. Jobs with higher priority scores are scheduled first.

#### **How Priority Scheduling Works:**

* Each job is assigned a **priority score** based on several factors, such as:
  + **Job age**: Older jobs typically have higher priority.
  + **User or group** priority: Admins can assign higher priorities to certain users or groups.
  + **Job size**: Larger jobs (e.g., requiring more CPUs) may be given lower priority in favor of smaller jobs that can be scheduled more quickly.
  + **Fair share**: Jobs are prioritized based on historical usage of resources by the user or group (fair share scheduling).

#### **Example Workflow:**

1. **Job A** is submitted and has a higher priority due to a combination of age and job size.
2. **Job B**, which was submitted later, has a lower priority and will be scheduled after Job A, even if Job A’s resources are not immediately available.

#### **Priority Scheduling Configuration:**

Priority-based scheduling in SLURM can be configured in the slurm.conf file using the following parameters:

PriorityType=priority/basic # Basic priority scheduling

PriorityWeightAge=1000 # Priority weight based on job age

PriorityWeightFairshare=500 # Priority weight based on fair share

PriorityWeightJobSize=500 # Priority weight based on job size

* **Job Age**: The older a job is, the higher its priority.
* **Fair Share**: Jobs from users or groups with less historical resource usage are prioritized.

### **3. Fair Share Scheduling Algorithm**

**Fair Share Scheduling (FSS)** is a priority-based scheduling policy designed to ensure that resources are allocated fairly over time. It calculates a **fair share** score for each user or group based on their historical resource usage.

#### 

#### **How Fair Share Works:**

* **Fair Share Scores**: SLURM calculates a fair share score for each user or group. The more resources a user has consumed in the past, the lower their fair share score becomes.
* **Job Priority Adjustment**: Jobs are scheduled based on the **fair share score**, ensuring that users who have used fewer resources in the past are given higher priority to run jobs in the future.

#### **Example Workflow:**

1. **User A** has used a large amount of resources recently and has a low fair share score.
2. **User B** has used fewer resources and is assigned a higher fair share score.
3. When both users submit jobs, **User B’s job** will have a higher priority, and **User A’s job** will be delayed until there are available resources.

#### **Fair Share Scheduling Configuration:**

Fair share scheduling is enabled and configured in the slurm.conf file by using parameters like:

PriorityType=priority/fairshare # Enable Fair Share Scheduling

PriorityDecayHalfLife=7-00:00:00 # Decay half-life for priority score calculation

PriorityWeightFairshare=1000 # Weight of the fair share in the priority score

* **PriorityDecayHalfLife**: This defines the time it takes for a user’s fair share score to decay by half, which helps to keep the scheduling fair over time.

### **4. Multilevel Scheduling Algorithm**

In **multilevel scheduling**, jobs are divided into different classes or levels, with each class having its own set of policies and priorities. This can be used in environments where jobs are grouped by type, priority, or resource requirements.

#### **How Multilevel Scheduling Works:**

* **Levels**: Jobs are divided into multiple classes or levels (e.g., high priority, low priority, etc.), each with distinct scheduling rules.
* **Each Level Has Different Policies**: Higher priority jobs can be scheduled according to one algorithm (e.g., First-Come-First-Served), while lower priority jobs can be scheduled according to another.

#### 

#### **Example Workflow:**

1. **Level 1 (high priority)** jobs are scheduled first, even if they take a lot of resources.
2. **Level 2 (low priority)** jobs are scheduled second, possibly backfilling in gaps left by higher-priority jobs.

#### **Multilevel Scheduling Configuration:**

Multilevel scheduling can be set up in SLURM with SchedulerType and Partition parameters. It may also require defining multiple scheduling policies in the configuration.

SchedulerType=sched/multilevel

### **5. First-Come, First-Served (FCFS)**

The **First-Come, First-Served (FCFS)** scheduler simply schedules jobs in the order they are submitted, with no consideration for job priority or resource requirements. This is a simple and fair approach but may not be ideal in systems where job runtimes and resource requirements vary significantly.

#### **How FCFS Works:**

* Jobs are queued in the order they are submitted.
* No backfilling or prioritization is done — all jobs in the queue must wait for resources to become available.

#### **Example Workflow:**

1. **Job A** arrives before **Job B** and will be scheduled first, regardless of its size or resource needs.
2. **Job B** waits in the queue until **Job A** completes, even if **Job A** is a very large job.

#### **FCFS Configuration:**

FCFS can be specified as the scheduler in the slurm.conf file using:

SchedulerType=sched/firstcome

This is typically used in simpler environments where fairness and simplicity are prioritized over resource optimization.

### **6. Custom Scheduler Algorithms**

In addition to the built-in scheduling algorithms, SLURM allows administrators to implement **custom scheduling policies** by using plugins. These plugins can extend SLURM’s default scheduling capabilities to meet specific needs, such as prioritizing certain types of jobs (e.g., batch vs. interactive), integrating with external systems (e.g., cloud resources), or adding additional features like job profiling.

#### **Example of Custom Scheduler Configuration:**

You can define a custom scheduler by setting up the plugin in the slurm.conf file:

SchedulerType=sched/plugin

PluginDir=/path/to/custom/plugin

Custom plugins are typically written in C or Python and can be tailored to the specific needs of the organization.

### **Conclusion**

SLURM provides a variety of scheduling algorithms, each designed to meet specific resource allocation and scheduling needs. Here's a quick overview of the algorithms:

1. **Backfill Scheduling**: Optimizes resource utilization by filling gaps between jobs.
2. **Priority Scheduling**: Jobs are scheduled based on priority scores, which can be affected by job age, size, and fair share.
3. **Fair Share Scheduling**: Ensures fairness by prioritizing jobs from users or groups with lower historical resource usage.
4. **Multilevel Scheduling**: Allows different job classes to have distinct policies for scheduling.
5. **FCFS (First-Come, First-Served)**: Simple scheduling based on the order of job submission.
6. **Custom Scheduling**: Allows for custom scheduling policies using plugins.

By configuring the appropriate scheduler algorithm, SLURM ensures that jobs are managed efficiently, resources are allocated fairly, and the cluster is utilized optimally.

### **SLURM Accounting: Tracking and Managing Job Resource Usage**

SLURM (Simple Linux Utility for Resource Management) provides powerful accounting features to track and manage the usage of resources in a computing cluster. **SLURM Accounting** allows administrators and users to monitor and analyze how computational resources (like CPU, memory, and time) are being utilized by jobs. It helps in auditing, optimizing resource allocation, and ensuring fair usage across users and groups.

SLURM’s accounting system stores detailed information about jobs, users, and their resource usage. This data can be used for billing purposes, performance analysis, and troubleshooting.

### **1. SLURM Accounting Overview**

SLURM’s accounting system is built around the **Accounting Storage** (commonly referred to as **slurmdbd**, or \*\*SLURM Database Daemon). The database stores all the job accounting records, which are collected by the **sacct** command or viewed using **sreport** for reporting. The accounting system provides valuable insights into job execution, resource consumption, and system usage patterns.

Accounting in SLURM is based on the **slurmdbd** (SLURM Database Daemon) which interacts with an external relational database system (typically MySQL or MariaDB) to store the data.

### **2. SLURM Accounting Configuration**

To use SLURM's accounting features, you need to configure SLURM to collect and store job usage data in a database. This is done by setting up the slurmdbd service and linking it to a relational database.

#### **a. slurm.conf Configuration**

You need to define accounting parameters in the **slurm.conf** file, which is the main configuration file for SLURM.

Example SLURM accounting-related configuration in slurm.conf:

# Enable accounting

AccountingStorageType=accounting\_storage/slurmdbd

AccountingStorageHost=slurmdbd\_host\_address # Database server hostname or IP address

AccountingStoragePort=6819 # Default port for SLURMDBD

AccountingStorageUser=slurmdbd\_user # Database user

AccountingStoragePass=slurmdbd\_password # Database password

* **AccountingStorageType**: Specifies that SLURM should use **slurmdbd** for accounting storage.
* **AccountingStorageHost**: Defines the database server where the accounting data is stored.
* **AccountingStoragePort**: Defines the port number used for SLURMDBD communication (default is 6819).
* **AccountingStorageUser** and **AccountingStoragePass**: Credentials to connect to the database.

Once the slurm.conf is configured, you also need to set up the slurmdbd service that will handle job records.

#### **b. slurmdbd.conf Configuration**

slurmdbd.conf is the configuration file for the SLURM Database Daemon (slurmdbd), which communicates with the accounting database (usually MySQL or MariaDB).

Example configuration:

# Define the database connection

DbdAddr=slurmdbd\_host\_address

DbdPort=6819

DbdUser=slurmdbd\_user

DbdPass=slurmdbd\_password

# Define authentication settings for slurmdbd (optional)

AuthType=auth/munge

Once these configurations are set, restart SLURM and the database daemon services.

### **3. SLURM Accounting Commands**

SLURM provides a few key commands for interacting with the accounting system and managing job resource usage data:

#### **a. sacct - Job Accounting Information**

The sacct command allows users to view historical job accounting information, such as CPU time, memory usage, and the state of jobs that have completed.

Example usage:

sacct --job=12345

This command will display accounting information for job ID 12345, including job resources (CPU, memory), elapsed time, exit status, etc.

Example output:

JobID JobName Partition Account AllocCPUs State ExitCode

---------- ---------- ---------- --------- ---------- ------ --------

12345 test\_job compute default 4 COMPLETED 0:0

* **JobID**: Unique identifier for the job.
* **JobName**: The name of the job.
* **Partition**: The SLURM partition the job ran in.
* **Account**: The account associated with the job.
* **AllocCPUs**: Number of CPUs allocated for the job.
* **State**: Job status (e.g., COMPLETED, FAILED, etc.).
* **ExitCode**: The job’s exit code.

#### **b. sreport - Reporting Job Accounting Data**

The sreport command is used to generate more comprehensive reports from the SLURM accounting database. This command allows administrators to create summaries of resource usage by user, account, or cluster.

Example usage:

sreport cluster utilization

This generates a report summarizing the utilization of resources (CPU, memory) across the entire cluster.

Example output:

ClusterName Nodes Cores CPUTime MemoryUsage ...

------------ ----- ----- ------- ------------ ...

cluster1 100 400 1200000 100GB …

#### **c. sacctmgr - Managing Accounting Data**

The sacctmgr command is used for managing accounts, users, and resources in the SLURM accounting system. It allows administrators to add or modify accounts, set resource limits, and assign jobs to accounts.

Example of adding a new account:

sacctmgr add account test\_account

Example of assigning a user to an account:

sacctmgr add user test\_user account=test\_account

#### **d. sstat - Display Running Job Statistics**

While sacct provides information about completed jobs, sstat can be used to display real-time statistics for jobs that are currently running.

Example usage:

sstat --job=12345

This command shows statistics for job 12345, such as current CPU usage and memory consumption.

Example output:

JobID MaxRSS MaxVMSize AveCPU AveDiskRead ...

------ --------- ------------ --------- ------------ ...

12345 1.5G 2.0G 75.5% 500MB

* **MaxRSS**: Maximum resident set size (maximum memory used).
* **MaxVMSize**: Maximum virtual memory size.
* **AveCPU**: Average CPU usage percentage.

**4. SLURM Accounting Data Structure**

SLURM stores job accounting data in a relational database. The data includes detailed records of resource consumption, job status, and other attributes that can be queried for analysis. Some key fields in job accounting records include:

* **JobID**: Unique job identifier.
* **User**: The user who submitted the job.
* **Account**: The user’s account for accounting purposes.
* **Job State**: Indicates the job’s state, such as COMPLETED, FAILED, or CANCELLED.
* **CPU Usage**: The total CPU time consumed by the job.
* **Memory Usage**: The maximum memory consumed by the job.
* **Walltime**: The total time the job was running.
* **Exit Status**: The exit code or completion status of the job.

### **5. Practical Applications of SLURM Accounting**

SLURM accounting data can be used for various purposes:

#### **a. Resource Usage Monitoring**

SLURM accounting allows users and administrators to monitor and review how computational resources are being used over time. This helps in identifying bottlenecks, under-utilized resources, or jobs consuming excessive resources.

#### **b. Cost Allocation and Billing**

SLURM accounting can track resource usage per user, project, or department. This data can be used for cost allocation, internal billing, or ensuring that resource usage aligns with budgetary constraints in multi-tenant environments.

#### **c. Fair Resource Distribution**

Using the accounting data, administrators can implement **Fair Share Scheduling** to allocate resources fairly among users or projects. By tracking how much resource a user has consumed historically, SLURM can adjust job priorities to ensure that no single user monopolizes the cluster.

#### **d. Performance Analysis**

SLURM accounting allows administrators to analyze job performance, such as job completion times, resource usage, and efficiency. This can guide decisions about cluster optimization, such as adding more resources or reconfiguring existing nodes.

### **6. SLURM Accounting Best Practices**

1. **Regular Database Backups**: As accounting data can grow over time, it is important to back up the accounting database regularly to prevent data loss.
2. **Data Retention Policies**: Set up retention policies for accounting data to manage storage. For instance, older job records may be archived or deleted after a certain period.
3. **Integrating SLURM with Billing Systems**: For clusters that charge users based on resource consumption, SLURM's accounting data can be integrated with billing systems to automate charge calculations.
4. **Utilizing Reports for Cluster Management**: Regularly generate and review reports using sreport to identify trends in resource usage and detect any issues, such as resource wastage or inefficient job execution.

### **Conclusion**

SLURM accounting provides a robust system for tracking and managing job resource usage, helping administrators optimize cluster performance, ensure fairness, and monitor resource consumption. By setting up SLURM's accounting system and using tools like sacct, sreport, and sacctmgr, users and administrators can gain valuable insights into cluster usage, manage costs, and improve resource allocation policies. The accounting system is a crucial part of large-scale cluster management, allowing for better tracking, reporting, and control of job execution.