IMAGENET ON OSG

Dataset used to benchmark:

Imagenet: (150 GB) **-** The ImageNet dataset is used as a benchmark because its 1.2 million images across 1,000 categories test a model's ability to generalize widely. Its challenge lies in the high variability and large number of categories, making it a robust indicator of a model's capability to handle complex, real-world data.

Initially, a tar file was used instead of a zip file of the data.

The ILSVRC2012_img_train.tar is used. **Structure of ILSVRC2012_img_train.tar:**

- **File Type:** `tar` archive containing the training images.
- **Content:** The tar file includes a directory structure where images are organized into folders corresponding to different classes.
- **Structure:**
  - **Root Directory:** The tar file typically contains a single root directory named `ILSVRC2012_img_train`.
  - **Class Subdirectories:** Inside the root directory, there are subdirectories named after each class label (e.g., `n01440764`, `n01443537`, etc.). Each subdirectory contains images belonging to that specific class.
- **Size:** The size of `ILSVRC2012_img_train.tar` is approximately **137 GB**. This large size reflects the extensive number of images included in the training set.

Implementation:

Training models on the ImageNet dataset presents significant challenges due to the dataset's large size and complexity. To efficiently utilize computational resources on the Open Science Grid (OSG), effective strategies for managing and processing this large dataset are crucial. The goal is to benchmark various methods for handling the dataset, evaluating their efficiency, resource usage, and overall feasibility.

In this context, three methods have been explored for training ImageNet on OSG:

1. **Method 1: Traditional Download and Extraction with Local Handling** - This approach involves downloading the dataset directly to the compute nodes, extracting it locally, and then using it for model training. This method is straightforward but requires substantial local storage and can be constrained by HTCondor's file transfer limits.
2. **Method 2: Direct Fetching from the Origin with HTTP** - This method simplifies the process by fetching the dataset directly from the origin via HTTPS within the job script. It integrates dataset retrieval, extraction, training, and cleanup into a single script, reducing the need for pre-job data management.

3. **Method 3: Direct Fetching from the Origin Using OSDF Protocol** - This approach leverages the OSDF protocol to fetch the dataset directly, streamlining data handling and reducing local storage requirements. It is designed to efficiently manage large datasets by utilizing the capabilities of OSDF for better data transfer and access.

Each method offers a different approach to managing the dataset, and their effectiveness in training ImageNet on OSG will be evaluated based on their resource utilization, efficiency, and practicality.

Python script explanation:

- This Python script is designed for training a ResNet50 model on an ImageNet-like dataset using PyTorch. It begins by importing essential libraries, including `argparse` for handling command-line arguments, `torch` for deep learning operations, `torch.nn` for neural network components, and `torch.optim` for optimization algorithms. Additionally, it utilizes `torchvision` for image transformations and pre-trained models, `os` for file handling, and `PIL` (Python Imaging Library) for image processing.
- The script defines a custom dataset class, `CustomImageNetDataset`, which extends PyTorch's `Dataset` class. This class is responsible for loading and preprocessing images from a directory structure similar to ImageNet. The constructor initializes the dataset by traversing directories to collect image paths and corresponding labels. Labels are converted into integer values for model training. The `__getitem__` method loads an image, applies any specified transformations, and returns the image along with its label.
- A set of image transformations is defined to preprocess images before feeding them into the model. These transformations include resizing, horizontal flipping, tensor conversion, and normalization, which are crucial for preparing the data in a format suitable for the ResNet50 model.
- The `ResNet50` class is defined, which wraps around PyTorch's pretrained ResNet50 model. It modifies the final fully connected layer to match the number of output classes (1000 for ImageNet). This adjustment ensures that the model can make predictions for the specific number of classes in the dataset.
- Training the model involves a `train` function that iterates over batches of data, computes the loss, performs backpropagation, and updates the model parameters using stochastic gradient descent (SGD). During training, it logs the loss and progress.
- The `main` function sets up argument parsing to handle command-line inputs, such as batch size, number of epochs, learning rate, and momentum. It initializes the dataset and data loader, sets the device to GPU if available, and creates an instance of the `ResNet50` model. The optimizer is configured with SGD and the model is trained over the specified number of epochs.
- Finally, if the `--save-model` argument is provided, the trained model's state dictionary is saved to a file named `imagenet_resnet50.pt`. This saved model can later be loaded for inference or further training.

<u>**METHOD 1:**</u>

Introduction:
Method 1 involves a straightforward approach where the dataset is downloaded, extracted locally on the compute nodes, and then used for training the model before cleaning up the temporary files. A python script is designed to download a large tar.gz file from a specified URL. This method is simple and easy to implement, making it convenient for straightforward data handling. However, it has significant drawbacks, including the need for substantial local storage to accommodate the dataset during download and extraction. This requirement can be problematic in environments with limited storage capacity or when dealing with very large datasets.

Process:
In this method, a single large zip file containing the entire dataset was used for training ImageNet on OSG. The process begins with the download of the ImageNet dataset, which is provided as a single large zip file named ILSVRC.zip. Once the file is downloaded, it is extracted locally on the compute nodes. The extraction process involves unzipping the file into a designated directory on the node. This approach ensures that the entire dataset is readily available in a structured format compatible with PyTorch's ImageFolder class.

During the training phase, a ResNet-50 model was utilized. The model was initialized from scratch, without using pre-trained weights. The training was conducted over 20 epochs, with model checkpoints saved at the end of each epoch to capture the state of the model at various stages of training.
After the training process was completed, the extracted data and the original zip file were removed from the compute nodes to free up storage space and maintain a clean working environment.

Result:
During the implementation of Method 1, an error was encountered due to the size of the input files exceeding the maximum allowed transfer limit configured in HTCondor. Specifically, the job was held with the following error message:

```
2024-07-18 14:35:31 Job was held.
    TransferInputSizeMB (158783) is greater than MAX_TRANSFER_INPUT_MB
(5000) at submit time
    Code 32 Subcode 0
```

This error indicates that the total size of the input files (approximately 150 GB) surpassed the maximum transfer limit of 5000 MB (5 GB). HTCondor imposes this limit to manage resource usage and ensure efficient job scheduling, which led to the job being held.

Proposed Solution - Data Splitting: The method involves splitting the large dataset into smaller, manageable chunks (each 5GB or less) to comply with transfer limits. The training script then processes each chunk sequentially, ensuring that the model is trained on the entire dataset over time. A single HTCondor job is submitted to handle transferring the dataset chunks and running the training script. During execution, the model is trained on each data chunk one by one, updating its state after processing each chunk. This allows for effective training on the entire dataset without the need for parallel processing, ensuring compliance with transfer limits and efficient use of resources.

Result of the Proposed Solution (Data Splitting):

Even though the dataset was divided into chunks, the total size of the input files reported was still above the 5 GB threshold. This is because HTCondor has a hard limit of 5 GB for the maximum input size per job, and it cannot handle input files exceeding this limit, regardless of how they are split.

Conclusion for Method 1:

Method 1, which involved using a single large zip file for training ImageNet on OSG, ultimately proved **unsuccessful.** Despite efforts to split the dataset into smaller chunks and downloading the file and extracting it locally on the compute nodes, the process was hindered by HTCondor's maximum input size constraint.

The total size of the dataset consistently exceeded HTCondor’s maximum allowed transfer limit of 5 GB. This limitation, imposed to manage resource usage and job scheduling, prevented the successful transfer of the dataset. As a result, both the initial approach of using a single large zip file and the attempt to handle the dataset by extracting it locally on the compute nodes were unsuccessful.

METHOD 2:

Introduction:
Method 2 involves leveraging the Open Science Data Federation (OSDF) protocol to directly fetch and process datasets for model training. This approach streamlines data handling by accessing the dataset directly from OSDF via HTTP, bypassing the need for manual downloading and extraction of large tarball files. The workflow for Method 2 includes direct access to the dataset, executing the model training process, and performing cleanup operations, all managed through HTCondor for job submission and resource allocation. This method simplifies data management, minimizes local storage requirements, and ensures efficient handling of large datasets, making it ideal for environments where effective data handling is critical.

Process:

1. **Data Fetching:** The dataset is accessed directly from the provided HTTPS URL: (transfer_input_files=`https://sdsc-origin.nationalresearchplatform.org:8443/nrp/osdf/ILSVRC2012/ILSVRC2012_img_train.tar`). This approach uses the HTTPS protocol for secure data transfer, avoiding the need for manual downloading or extraction of tarballs.
2. **Wrapper Script Execution:**
   - The wrapper script sets up the environment and prepares the dataset for training. It begins by creating a directory named `data` and then moves the `ILSVRC2012_img_train.tar` file into this directory. The script then extracts the tar file containing the ImageNet training data.
   - After extracting the primary tar file, the script handles any additional tar files found within. It creates directories for each class and extracts the images into their respective directories. To keep the logs clean, the extraction output is suppressed. Finally, the script removes the tar files after extraction to save space and ensure that only the necessary files remain.
3. **Model Training:** With the data prepared, the script executes the Python training script `main2.py`. This script trains the ResNet50 model using PyTorch, with the training configured to save the model after the specified number of epochs. The use of PyTorch and the Singularity image ensures compatibility and efficient execution.
4. **Cleanup:** After the training process is complete, the wrapper script cleans up by removing the `data` directory and all its contents. This step ensures that no residual files are left on the compute nodes, maintaining a clean environment and freeing up storage resources.

Result:
During the implementation of Method 2, an error was encountered. The transfer failed with an error indicating a lack of progress, which led to job eviction and holding. This issue is likely due to network problems, server unresponsiveness, or transfer interruptions.

```
012 (791469.000.000) 2024-08-09 22:31:42 Job was held.
	Transfer input files failure at the execution point using protocol https. Details: Aborted
due to lack of progress
```

The file transfer was interrupted because it was not making adequate progress. The transfer from the specified URL to the job's execution node failed, possibly due to network issues, slow download speeds, or server-side problems. The system aborted the transfer after it determined that progress was insufficient, preventing the job from starting successfully.

During an attempt to rerun, the same error occurred. Further using wget in the wrapper script was tried.

Proposed Solution - wget command in wrapper script - In this method, the wrapper script is configured to use `wget` to fetch the ImageNet dataset directly from OSDF via HTTP. By specifying the URL, the script automates the process of downloading the dataset during job execution.

Result of the Proposed Solution:

```
/srv//imagenet-2.sh: line 6: wget: command not found/srv//imagenet-2.sh: line 6: wget: command not found
```

The error indicates that the wget command was not available in the environment where the job was running. This means that wget was either not installed or not included in the system's executable path, preventing the download of the dataset.

Conclusion for Method 2:

Method 2, which aimed to streamline dataset handling by directly fetching from OSDF via HTTPS, encountered significant challenges and was ultimately unsuccessful. Initially, file transfer issues led to job eviction due to lack of progress, likely caused by network problems or server issues. Attempts to resolve this by incorporating `wget` in the wrapper script failed as the command was not available in the execution environment. This method's efficiency in managing large datasets through direct HTTP access is compromised by these technical difficulties, highlighting the need for reliable data transfer mechanisms and availability of required tools in the job environment. Further investigation and alternative solutions are necessary to overcome these issues and achieve successful implementation.

METHOD 3:

Introduction:
Method 3 involves fetching the dataset directly from the origin using the Open Science Data Framework (OSDF) protocol. This approach is designed to streamline data handling by leveraging OSDF's capabilities for efficient data transfer and access. The dataset is accessed through OSDF during the job execution, which reduces the need for substantial local storage and minimizes the overhead associated with data management. By integrating dataset retrieval directly within the job script, Method 3 simplifies the workflow and ensures that large datasets can be handled effectively without overwhelming local storage resources. This method is particularly advantageous in environments where managing large volumes of data efficiently is critical.

Process:

Method 3 involves leveraging the Open Science Data Framework (OSDF) protocol to efficiently manage and process the ImageNet dataset during model training. This approach is designed to streamline data handling by accessing the dataset directly from OSDF, thereby reducing the need for extensive local storage and simplifying the workflow.

Detailed Steps:

1. **Data Fetching:** The dataset is accessed from OSDF, specifically from the path `osdf:///nrp/osdf/ILSVRC2012/ILSVRC2012_img_train.tar`. This method utilizes OSDF's efficient data transfer capabilities, which are designed to handle large datasets and minimize the overhead of manual data management.
2. **Wrapper Script Execution:** The wrapper script is responsible for setting up the environment, managing the dataset, and executing the training process. Initially, it creates a directory named `data` and moves the `ILSVRC2012_img_train.tar` file into this directory. The script then extracts the tar file, which contains the ImageNet training data.
   After extracting the primary tar file, the script proceeds to handle individual class tar files contained within. It creates directories for each class and extracts the images into their respective directories. This process involves suppressing the output of the extraction commands to avoid cluttering the logs. Finally, the script removes the tar files after extraction to save space and ensure that only the necessary files remain.
3. **Model Training:** With the data prepared, the script invokes the Python training script `main3.py`, which handles the training of the ResNet50 model. The training script is configured to save the trained model after completing the specified number of epochs. The model is trained using the PyTorch library, which is compatible with the Singularity image specified for the job.
4. **Cleanup:** After training is complete, the wrapper script performs cleanup by removing the `data` directory and all its contents. This ensures that no residual files are left on the compute nodes, maintaining a clean environment and freeing up storage resources

Errors:

1. SciPy Module Installation Error

During the job execution, an `OSError: [Errno 38] Function not implemented` was encountered while installing the SciPy module. This error typically occurs due to limitations or restrictions in the filesystem or environment permissions, which might prevent certain operations from being performed. Additionally, a `ModuleNotFoundError` indicated that the SciPy module was not available when the script attempted to use it. To address this, manual installation of the SciPy module using `pip install scipy was done`. This action resolved the missing module issue, allowing the job to continue its execution without encountering further errors related to SciPy.

2. Failed File Transfer (404 Error)

The job initially failed to transfer a file from the URL `osdf:///ospool/imagenet/imagenet-object-localization-challenge.zip`, resulting in a 404 error. This error indicates that the file or the specified path could not be found on the storage system. Such issues can arise from incorrect file paths, non-existent files, or misconfigured storage systems. After identifying the problem, the transfer process was retried, which was successful on subsequent attempts. This allowed the necessary file to be transferred and made available for the job, resolving the initial file transfer failure.

3. Zip File Handling Failure

The job encountered issues with handling a zip file due to the absence of the unzip command in the execution environment. This problem prevented the extraction of data from the zip file, which was crucial for the job. To overcome this issue,the problematic zip file was replaced with a tar file. The tar format was successfully processed and extracted without encountering the same issues, thus enabling the job to access and use the data as intended. This change resolved the difficulties related to zip file handling and ensured smooth data processing.

4. Archive Problem

A problem arose when the job attempted to access the dataset archive `ILSVRC2012_devkit_t12.tar.gz`. This file was missing, leading to runtime errors in the Python script that relied on it for data processing. The absence of this archive prevented the job from proceeding as expected. As a corrective measure,the dataset archive, `ILSVRC2012_img_train.tar` was continued to be used, which was available for training the model by creating a custom dataset class, `CustomImageNetDataset`, to handle the training data more effectively. This adjustment allowed the training process to move forward without being hindered by the corrupted or missing file.

5. Socket Connection Closure and Job Disconnection

The job experienced disconnection issues due to unexpected socket closures between the submit and execution hosts. This led to the job exceeding the 40-minute reconnection window, which is a common timeout limit in job scheduling systems. Network disruptions or problems with the execution host can cause such issues. To mitigate the risk of disconnections, the allocated disk space and other resources for the job were increased. This adjustment was aimed at providing a more stable environment and reducing the likelihood of similar disconnection problems in future job executions.

6. Repeated Disconnections and Job Lease Expiration

The job faced multiple disconnections, which led to lease expiration and eventual eviction. Lease expiration occurs when a job cannot maintain a connection within the allocated time frame, resulting in its eviction from the queue. This problem is often due to persistent network or system failures. To address this issue, the allocated disk space and other resources were increased for the job, which helped to stabilize the environment and minimize the likelihood of further disconnections. These adjustments improved the job's stability, allowing it to complete successfully in subsequent runs.

7. Persistent File Transfer Stalls and Job Evictions

During the job execution, there were repeated stalls in the file transfer process, leading to multiple job evictions. The stalls in file transfers and subsequent job evictions were primarily due to an overloaded cache, which impeded the efficient handling of large files. When the cache becomes overloaded, it can cause significant delays in file transfer operations, leading to interruptions in job execution. Network instability may also exacerbate these issues, but the primary cause in this case was the cache overload. Once the cache-related issues were addressed, the job was able to execute without further errors. They still stall sometimes now but the file transfer is successful after some tries.

Result:
The ResNet model was successfully trained on the ImageNet dataset for 20 epochs, with the entire process taking approximately 14 hours. The training was completed on the resource `GLIDEIN_ResourceName = "GP-ARGO-ksu-backfill"`. After overcoming the initial file transfer and execution issues, the job was executed without further errors and finished successfully.

Subsequently, the training process was tested by queuing the job five times for 20 epochs. These jobs were distributed across different sites, demonstrating the model's capability to run consistently across multiple resources. The tests validated the robustness and portability of the training setup, confirming successful execution in diverse environments.

<u>For 20 epochs:</u>

JOB ID	GLIDEIN_Resource Name	TimeExecute (s)	TimeSlotBusy (s)	Time for transferring input files (total)
791459.000.000	PDX-Coeus-CE1	56290	57782	7 hours, 42 min, and 26 sec (file transfer stalled 7 times)
791459.001.000	Stuck job, kept stalling. It was removed later.			
791459.002.000	CHTC-Spark-CE1	43339	46489	53 min and 28 sec
791459.003.000	GP-ARGO-ksu-backfill	64328	65174	14 min and 2 sec
791459.004.000	PDX-Coeus-CE1	56057	57795	5 hours, 56 min, and 1 sec (file transfer stalled thrice)

The job, which involved training a ResNet50 model on the ImageNet dataset, was successfully executed across several sites. The following observations and results were noted:

1. **Successful Executions:**
    - Jobs running on `PDX-Coeus-CE1`, `CHTC-Spark-CE1`, and `GP-ARGO-ksu-backfill` were completed successfully. Each site demonstrated varying performance in terms of file transfer and overall execution time.
2. **Performance Insights:**
    - **Least Time for Transferring Input Files:** The job executed on `GP-ARGO-ksu-backfill` exhibited the shortest file transfer time, taking only 14 minutes and 2 seconds. This indicates efficient handling of file transfers at this site.
    - **Least Time to Execute:** The job on `CHTC-Spark-CE1` had the shortest overall execution time, completing in 12 hours. This reflects efficient processing and resource utilization during model training.
3. **Overall Efficiency:**
    - The job's success across different sites highlights the importance of both effective file transfer and computational efficiency. `GP-ARGO-ksu-backfill` was notably efficient in file transfers, while `CHTC-Spark-CE1` achieved the best overall execution time.
    - Additionally, it was observed that when input files are transferred for the second time on the same site, the process takes significantly less time, leading to faster results. This indicates that once a site has preloaded the necessary data, subsequent jobs benefit from reduced transfer times, further enhancing overall job efficiency.

<u>For 5 epochs:</u>

JOB ID	**GLIDEIN_ Resource Name**	**TimeExecute (s)**	**TimeSlotBusy (s)**	**Time for transferring input files (total)**
794421.000.000	CHTC-Spark-CE1	10209	10932	12 min 1 sec
794421.001.000	GP-ARGO-ksu-backfill	16757	17539	12 min 58 sec
794421.002.000	CHTC-Spark-CE1	11039	11533	1 hour 8 min 41 sec (stalled once); last transfer- 8 min 12 sec
794421.003.000	GP-ARGO-ksu-backfill	17424	18202	2 hours 52 min 39 sec (stalled once) ; last transfer- 12 min 53 sec
794421.004.000	CHTC-Spark-CE1	10407	11132	12 min 3 sec

The jobs executed across CHTC-Spark-CE1 and GP-ARGO-ksu-backfill sites yielded important insights into the performance and efficiency of the model's training process.

1. **Consistency in Execution Times:** Jobs run on CHTC-Spark-CE1 exhibited more consistent and slightly shorter execution times compared to GP-ARGO-ksu-backfill. Specifically, the job on CHTC-Spark-CE1 with Job ID 794421.000.000 completed in 10,209 seconds (approximately 2 hours 50 minutes), while GP-ARGO-ksu-backfill took 16,757 seconds (approximately 4 hours 40 minutes) for Job ID 794421.001.000.
2. **Impact of File Transfer Delays:** The transfer times varied significantly across different runs. The first job on GP-ARGO-ksu-backfill had a reasonable transfer time of 12 minutes 58 seconds, but a later job (794421.003.000) on the same site faced a substantial delay, with the transfer taking 2 hours 52 minutes and 39 seconds due to a stall. The same happened with CHTC-Spark-CE1.

Method 3 effectively utilized the Open Science Data Framework (OSDF) to streamline the ImageNet dataset handling and model training process.

Subsequent tests across different sites confirmed the method's robustness:

- **Shortest File Transfer Time**: GP-ARGO-ksu-backfill at 14 minutes and 2 seconds.
- **Fastest Overall Execution Time**: CHTC-Spark-CE1 with a 12-hour total runtime.
- **Failures (Stuck and Other Issues):** 20% (takes a lot of time to complete)
- **Success Rate:** 80%

These results suggest that selecting sites with efficient file handling capabilities and optimized computational resources can significantly impact job performance and completion time.

In conclusion for 5 epochs, while both CHTC-Spark-CE1 and GP-ARGO-ksu-backfill demonstrated their capabilities, the occurrence of stalling significantly impacted the efficiency of job executions. If stalling during file transfers can be avoided or minimized, the overall performance and efficiency of both sites could be greatly improved, resulting in more reliable and faster job completions.

For 5 epochs (100 jobs - submitted twice (queue 50):

JOB ID	GLIDEIN_ Resource Name	TimeExecute (s)	TimeSlot Busy (s)	Time for transferring input files (total)	Time for last transfer (if stalled)
794569.000.000	CHTC-Spark-CE1	11182	13283	35 min	-
794569.001.000	CHTC-Spark-CE1	11074	13177	35 min 2 sec	-
794569.002.000	PDX-Coeus-CE1	15117	17833	45 min 10 sec	-
794569.003.000	PDX-Coeus-CE1	15136	17872	45 min 29 sec	-
794569.004.000	PDX-Coeus-CE1	Execute: 978 SlotBusy: 3617 Input file transfer successful but no output as the network was unreachable		13 hours 6 min 9 sec (stalled 4 times)	43 min 55 sec
794569.005.000	GP-ARGO-ksu-backfill	Job was held. The job exceeded allowed execute duration of 20:00:00 Code 47 Subcode 0		13 min 19 sec	-
794569.006.000	PDX-Coeus-CE1	Execute: 977 SlotBusy: 3641 Input file transfer successful but no output as the network was unreachable.		11 hours 43 min 52 sec (stalled 3 times)	44 min 19 sec
794569.007.000	CHTC-Spark-CE1	11076	14357	54 min 39 sec	-
794569.008.000	CHTC-Spark-CE1	11232	13829	43 min 15 sec	-
794569.009.000	PDX-Coeus-CE1	14857	17558	44 min 55 sec	-
794569.010.000	PDX-Coeus-CE1	14892	17564	44 min 27 sec	-
794569.011.000	CHTC-Spark-CE1	10993	13638	14 hours 48 min 8 sec (stalled 13 times)	44 min 4 sec

794569.012.000	PDX-Coeus-CE1	14856	17850	8 hours 8 min 5 sec (stalled 7 times)	49 min 48 sec
794569.013.000	CHTC-Spark-CE1	11506	13782	14 hours 5 min 58 sec (stalled 13 times)	37 min 55 sec
794569.014.000	CHTC-Spark-CE1	Output file was transferred but the model was not trained. Errors like not valid tar file and empty dataset (but the same tar file was used for all jobs)		14 hours 48 min 22 sec (first file transfer- 5 hours 24 min 30 sec - stalled 5 times but job was held. Tried again - 8 hours 42 min 33 sec - stalled 7 times - successful)	41 min 54 sec
794569.015.000	CHTC-Spark-CE1	11323	11751	18 hours 34 min 45 sec (stalled 14 times)	7 min 7 sec
794569.016.000	PDX-Coeus-CE1	14837	17726	48 min 3 sec	-
794569.017.000	PDX-Coeus-CE1	14844	17829	49 min 39 sec	-
794569.018.000	GP-ARGO-ksu-backfill	Job was held. The job exceeded allowed execute duration of 20:00:00 Code 47 Subcode 0		25 min 34 sec	-
794569.019.000	CHTC-Spark-CE1	11268	14662	14 hours 55 min 53 sec (stalled 6 times)	56 min 33 sec
794569.020.000	PDX-Coeus-CE1	14895	17842	49 min 1 sec	-
794569.021.000	PDX-Coeus-CE1	14909	17601	2 hours 44 min 19 sec (stalled 1 time)	44 min 46 sec
794569.022.000	PDX-Coeus-CE1	14945	17636	44 min 45 sec	-
794569.023.000	CHTC-Spark-CE1	11254	11722	7 min 46 sec	-
794569.024.000	GP-ARGO-ksu-backfill	13170	14348	15 hours 19 min 29 sec (stalled 13 times)	19 min 35 sec

794569.025.000	GP-ARGO-ksu-backfill	16325	17104	4 hours 18 min 55 sec (stalled 3 times)	12 min 55 sec
794569.026.000	CHTC-Spark-CE1	10887	13128	37 min 19 sec	-
794569.027.000	CHTC-Spark-CE1	10896	13157	37 min 38 sec	-
794569.028.000	PDX-Coeus-CE1	14893	17921	50 min 23 sec	-
794569.029.000	PDX-Coeus-CE1	14901	17931	50 min 24 sec	-
794569.030.000	CHTC-Spark-CE1	11292	11847	12 hours 24 min 9 sec (first file transfer - 8 hours 31 min 42 sec -stalled 8 times but the job was held. Tried again - 2 hours 12 min 51 sec - stalled 3 times- successful)	9 min 13 sec
794569.031.000	CHTC-Spark-CE1	10891	12045	1 hour 21 min 45 sec (stalled 1 time)	19 min 13 min
794569.032.000	CHTC-Spark-CE1	11475	12347	1 hour 17 min 9 sec (stalled 1 time)	14 min 30 sec
794569.033.000	CHTC-Spark-CE1	11242	11735	8 min 11 sec	-
794569.034.000	CHTC-Spark-CE1	11523	11993	17 hours 32 min 10 sec (first file transfer -14 hours 16 min 56 sec - stalled 12 times but the job was held. Tried again - 2 hours 12 min 20 sec stalled 2 times - successful)	7 min 49 sec
794569.035.000	PDX-Coeus-CE1	14854	17859	50 min	-
794569.036.000	PDX-Coeus-CE1	14871	17760	48 min 3 sec	-

794569.037.000	CHTC-Spark-CE1 GP-ARGO-ksu-backfill	17608	18437	19 hours 56 min 42 sec (first file transfer - 11 hours 43 min 4 sec - stalled 10 times but the job was held. Tried again - 6 hours 34 min 7 sec stalled 6 times but the job was held again. Tried again- 13 min 47 sec- successful)	13 min 47 sec
794569.038.000	CHTC-Spark-CE1	11430	11838	7 hours 33 min 33 sec (stalled 5 times)	6 min 47 sec
794569.039.000	GP-ARGO-ksu-backfill	18937	19919	16 min 14 sec	-
794569.040.000	CHTC-Spark-CE1	11426	11886	4 hours 45 min 50 sec (stalled 2 times)	7 min 39 sec
794569.041.000	CHTC-Spark-CE1	11634	13907	1 hour 40 min 50 sec (stalled 1 time)	37 min 52 sec
794569.042.000	GP-ARGO-ksu-backfill	31422	32597	19 min 32 sec	-
794569.043.000	CHTC-Spark-CE1	10653	13698	50 min 42 sec	-
794569.044.000	CHTC-Spark-CE1	11089	13751	44 min 20 sec	-
794569.045.000	CHTC-Spark-CE1	11051	12700	3 hours 24 min 53 sec (job was held once and stalled 1 time)	27 min 28 sec
794569.046.000	CHTC-Spark-CE1	11119	11721	3 hours 24 min 16 sec (job was held once)	10 min 1 sec
794569.047.000	PDX-Coeus-CE1	14851	17874	50 min 5 sec	-
794569.048.000	PDX-Coeus-CE1	14826	17845	50 min 14 sec	-
794569.049.000	CHTC-Spark-CE1	Job 794569.049.000 was held due to a 404 error when transferring input files via the osdf protocol.			

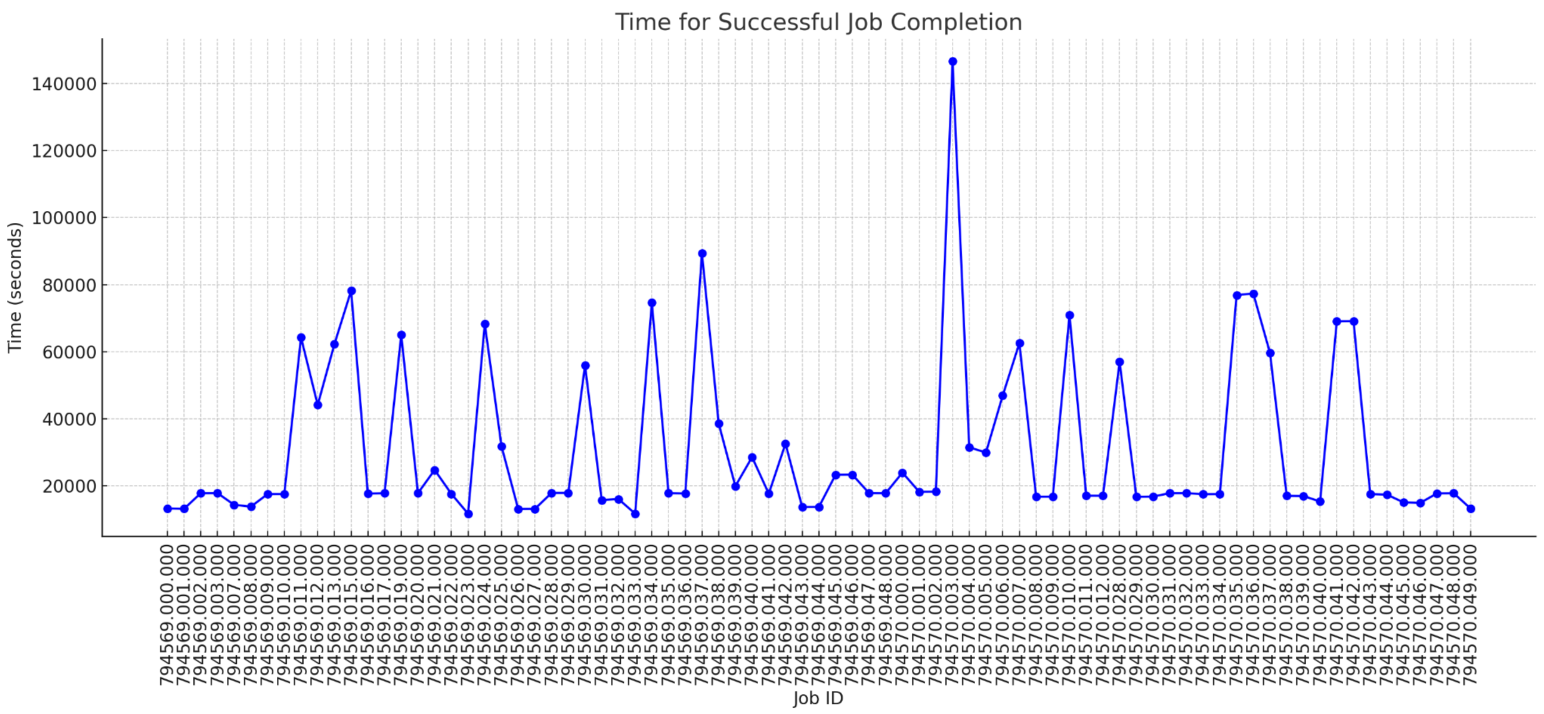
794570.000.000	CHTC-Spark-CE1	11819	12302	3 hours 22 min 34 sec (stalled 1 time)	8 min
794570.001.000	PDX-Coeus-CE1	14818	18265	57 min 21 sec	-
794570.002.000	PDX-Coeus-CE1	14853	18296	57 min 16 sec	-
794570.003.000	PDX-Coeus-CE1	14952	17592	36 hours 34 min 10 sec (stalled 28 times)	43 min 55 sec
794570.004.000	CHTC-Spark-CE1 PDX-Coeus-CE1	14843	17833	4 hours 37 min 37 sec (- 9 min 31 sec at CHTC-Spark-CE1 but the job was evicted. - 49 min 44 sec at PDX-Coeus-CE1)	49 min 44 sec
794570.005.000	CHTC-Spark-CE1 PDX-Coeus-CE1	14878	17834	4 hours 11 min 15 sec (- 8 min at CHTC-Spark-CE1 but the job was evicted. - 49 min 10 sec at PDX-Coeus-CE1)	49 min 10 sec
794570.006.000	CHTC-Spark-CE1 CHTC-Spark-CE1 GP-ARGO-ksu-backfill	24173	25017	6 hours 19 min 2 sec (- 7 min 27 sec at CHTC-Spark-CE1 but the job was evicted. - 8 min 42 sec at CHTC-Spark-CE1 but the job got evicted again. - 5 hours 16 min 6 sec at GP-ARGO-ksu-backfill where it stalled once)	14 min 1 sec
794570.007.000	GP-ARGO-ksu-backfill	19034	19965	12 hours 5 min 37 sec (stalled 1 time)	15 min 26 sec
794570.008.000	PDX-Coeus-CE1	14587	16796	36 min 44 sec	-

794570.009.000	PDX-Coeus-CE1	14603	16810	36 min 42 sec	-
794570.010.000	GP-ARGO-ksu-backfill	19746 49304	20691 50091	6 hours 42 sec (15 min 39 sec first file transfer but job got evicted)	13 min 4 sec
794570.011.000	PDX-Coeus-CE1	14673	17078	39 min 59 sec	-
794570.012.000	PDX-Coeus-CE1	14620	17072	40 min 45 sec	-
794570.013.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				
794570.014.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				
794570.015.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				
794570.016.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				
794570.017.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				
794570.018.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				
794570.019.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				
794570.020.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				
794570.021.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				
794570.022.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				
794570.023.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				
794570.024.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				
794570.025.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				
794570.026.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				

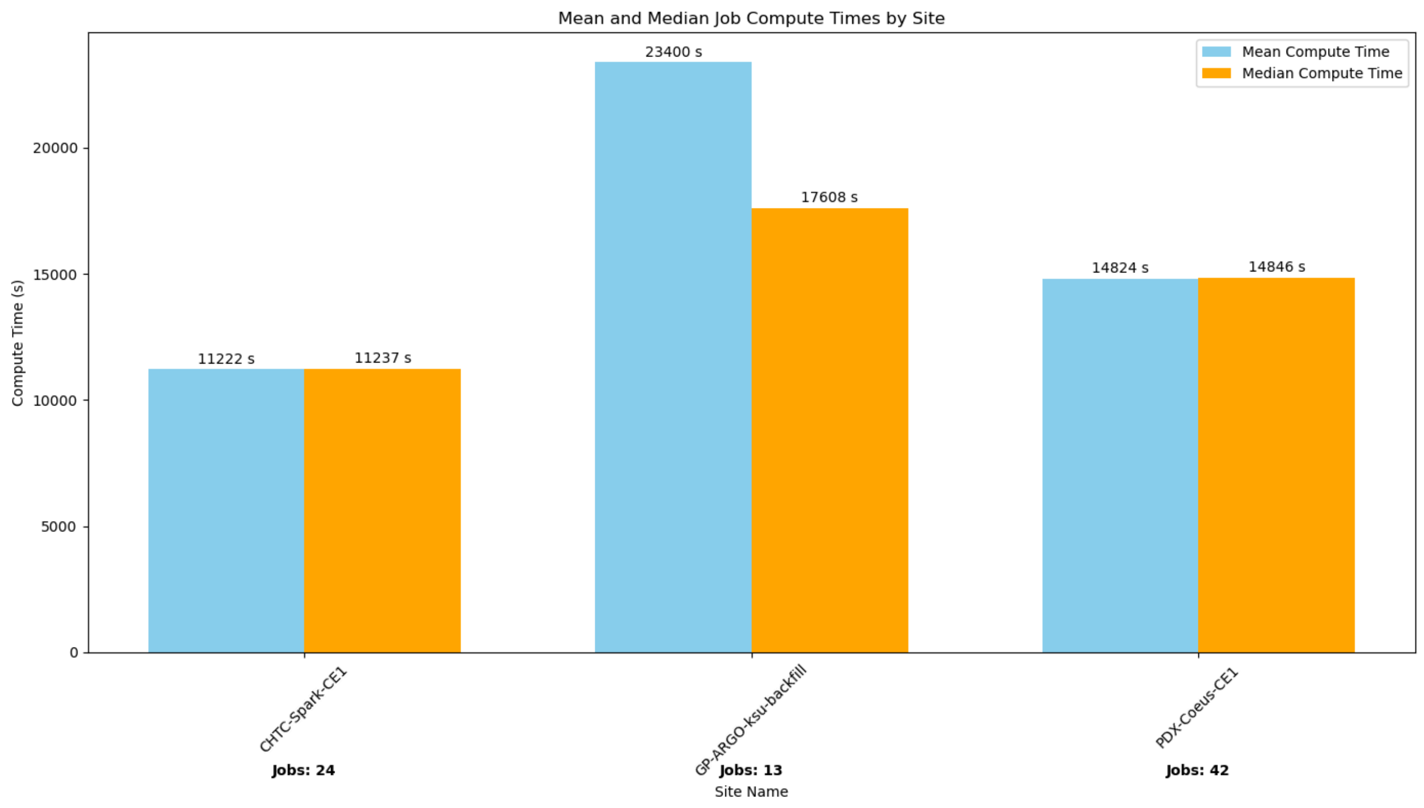
794570.027.000	Job was held due to a failure in transferring input files, with a "404 page not found" error.				
794570.028.000	PDX-Coeus-CE1	14967	17620	11 hours 40 min 10 sec (stalled 8 times)	44 min 8 sec
794570.029.000	PDX-Coeus-CE1	14609	16769	35 min 54 sec	-
794570.030.000	PDX-Coeus-CE1	14635	16828	36 min 28 sec	-
794570.031.000	PDX-Coeus-CE1	14815	17846	50 min 25 sec	-
794570.032.000	PDX-Coeus-CE1	14848	17867	50 min 14 sec	-
794570.033.000	PDX-Coeus-CE1	14835	17527	44 min 47 sec	-
794570.034.000	PDX-Coeus-CE1	14862	17560	44 min 52 sec	-
794570.035.000	PDX-Coeus-CE1	14726	16902	17 hours 16 min 10 sec (stalled 13 times)	36 min 11 sec
794570.036.000	PDX-Coeus-CE1	14659	16785	17 hours 25 min 20 sec (stalled 13 times)	35 min 20 sec
794570.037.000	GP-ARGO-ksu-backfill	58683	59669	16 min 23 sec	-
794570.038.000	PDX-Coeus-CE1	14621	17040	40 min 14 sec	-
794570.039.000	PDX-Coeus-CE1	14586	16991	39 min 59 sec	-
794570.040.000	GP-ARGO-ksu-backfill	14598	15362	12 min 37 sec	-
794570.041.000	PDX-Coeus-CE1	14821	17488	15 hours 4 min 20 sec (stalled 13 times)	44 min 20 sec
794570.042.000	PDX-Coeus-CE1	14841	17508	15 hours 4 min 20 sec (stalled 13 times)	44 min 21 sec
794570.043.000	PDX-Coeus-CE1	14947	17570	43 min 38 sec	-

794570.044.000	PDX-Coeus-CE1	14872	17407	41 min 56 sec	-
794570.045.000	GP-ARGO-ksu-backfill	14324	15088	12 min 40 sec	-
794570.046.000	GP-ARGO-ksu-backfill	14154	14965	13 min 27 sec	-
794570.047.000	PDX-Coeus-CE1	14808	17792	49 min 38 sec	-
794570.048.000	PDX-Coeus-CE1	14840	17827	49 min 41 sec	-
794570.049.000	GP-ARGO-ksu-backfill	12467	13300	13 min 51 sec	-

Time successful jobs took from first being matched to successful completion (starting file transfer to job termination) :



The line chart depicts the time taken for successful job completion from the start of file transfer to job termination across multiple job IDs. The completion times exhibit significant variability, with most jobs completing between 20,000 and 60,000 seconds. However, there are noticeable spikes, with some jobs taking over 100,000 seconds, indicating occasional outliers that experience much longer processing times. These outliers suggest variability in resource allocation or job complexity.

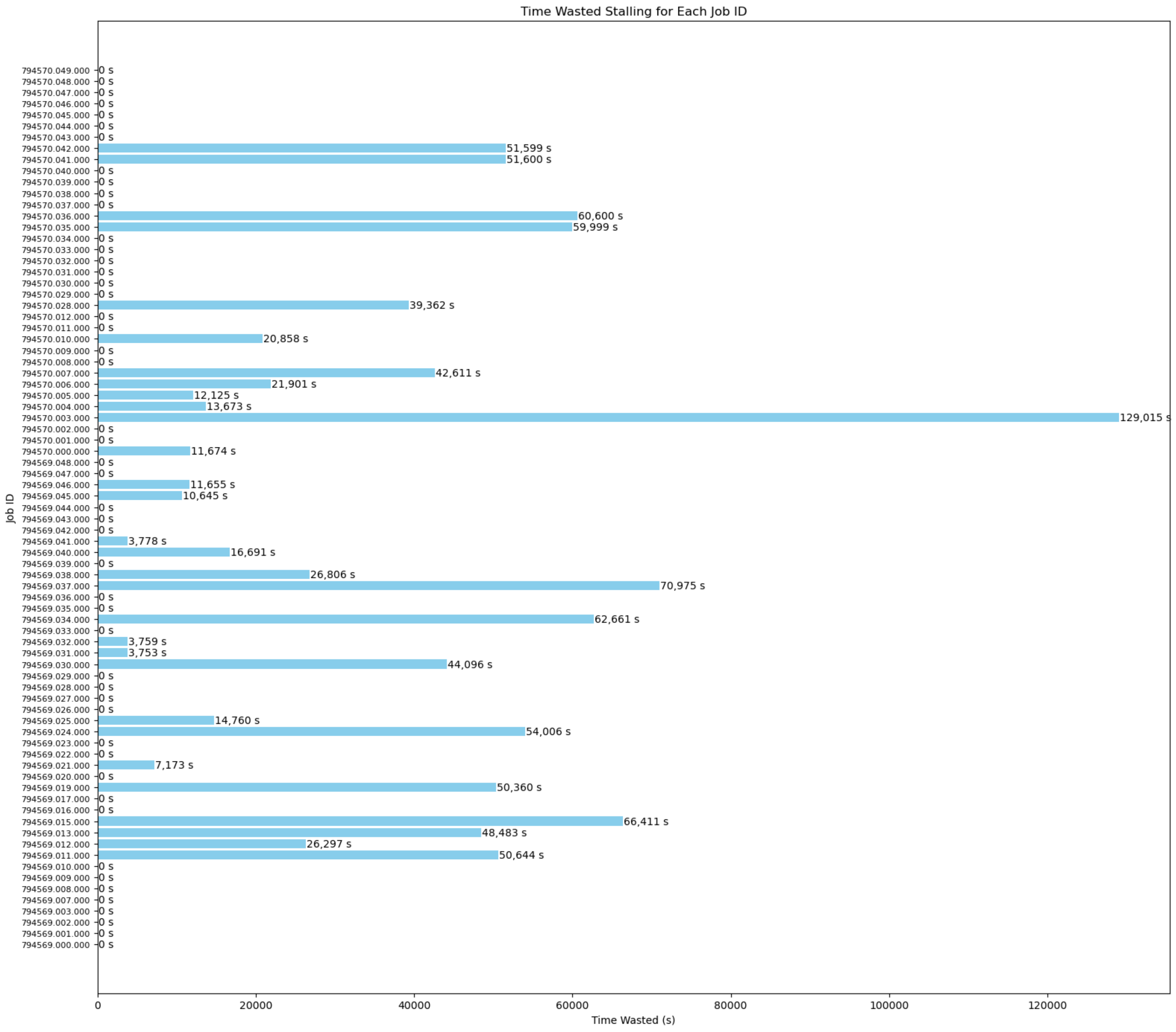


The bar chart provides a comparison of mean and median job compute times across three sites: CHTC-SpArK-CE1, GP-ARGO-Xsu-backfill, and PDX-Coeus-CE1.

- The data reveals that GP-ARGO-Xsu-backfill has significantly higher compute times, with a mean of 23,400 seconds and a median of 17,608 seconds, indicating a larger variability in job durations at this site.
- In contrast, CHTC-SpArK-CE1 exhibits the lowest compute times, with a mean of 11,222 seconds and a median of 11,237 seconds, suggesting a consistent job duration with minimal variability.
- PDX-Coeus-CE1 falls in the middle, with mean and median compute times close to each other, at 14,824 seconds and 14,846 seconds, respectively, indicating a consistent performance despite handling the largest number of jobs (42).

GP-ARGO-Xsu-backfill experiences higher compute times, possibly due to factors like resource availability or job complexity.

CHTC-SpArK-CE1 and PDX-Coeus-CE1 have more consistent and lower compute times, with PDX-Coeus-CE1 managing a larger workload effectively.



The bar chart illustrates the time wasted due to stalling for each job ID, showing significant variation across jobs. Some jobs experience minimal stalling, while others waste substantial time, with one job wasting over 129,000 seconds. The high variability in stalling times suggests inefficiencies in resource allocation or scheduling, leading to delays in job processing. This highlights the need for optimization to reduce wasted time and improve overall job efficiency.

Unsuccessful jobs:

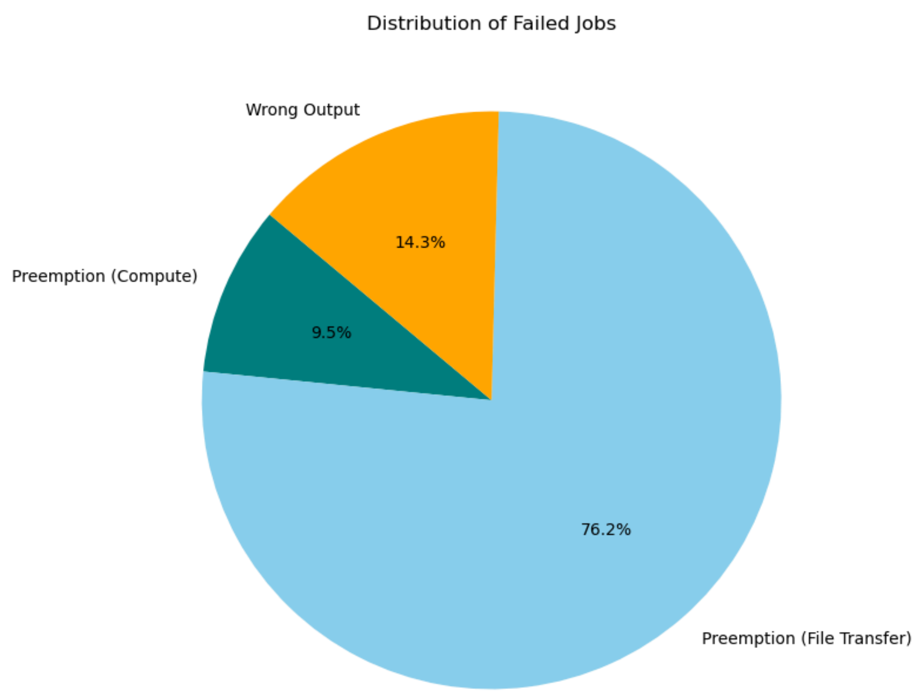
Failed jobs (wrong output):

JOB ID	**TimeExecute (s)**	**Time for last transfer (if stalled)**
794569.004.000	978	43 min 55 sec
794569.006.000	977	44 min 19 sec
794569.014.000	33	41 min 54 sec

Preempted failed job (job was held):

JOB ID	**Job Held during:**	**TimeExecute (s)**	**Time for last transfer (if stalled)**
794569.005.000	Compute	72024	13 min 19 sec
794569.018.000	Compute	72065	25 min 34 sec
794569.049.000	File transfer	-	1 sec
794570.013.000	File transfer	-	1 sec
794570.014.000	File transfer	-	1 sec
794570.015.000	File transfer	-	1 sec
794570.016.000	File transfer	-	1 sec
794570.017.000	File transfer	-	1 sec
794570.018.000	File transfer	-	1 sec
794570.019.000	File transfer	-	1 sec
794570.020.000	File transfer	-	1 sec
794570.021.000	File transfer	-	1 sec
794570.022.000	File transfer	-	1 sec
794570.023.000	File transfer	-	1 sec
794570.024.000	File transfer	-	1 sec
794570.025.000	File transfer	-	1 sec
794570.026.000	File transfer	-	1 sec

794570.027.000	File transfer	-	1 sec



The chart provides a detailed analysis of the reasons behind job failures within a particular system or process. The distribution is divided into three primary categories:

- Preemption (File Transfer): This category represents the largest share of failures, accounting for 76.2% of the unsuccessful jobs. Preemption here likely refers to interruptions or terminations of file transfer processes before they could be completed. This could be due to resource reallocation, system priorities, or other external factors that caused the file transfers to fail.
- Wrong Output: This is the second most common cause of failure, making up 14.3% of the unsuccessful jobs. Jobs categorized under "Wrong Output" are those that completed execution but produced incorrect or unexpected results, which would require reprocessing or debugging to correct.
- Preemption (Compute): This category, which accounts for 9.5% of the failures, involves jobs that were preempted during the computation phase. Similar to file transfer preemption, these interruptions could have been caused by resource shortages, higher-priority processes, or system errors that halted the computation mid-process.

The total number of failed jobs is 21, indicating that these 21 jobs did not complete successfully for the reasons mentioned above. Despite these failures, the overall success rate of the jobs stands at 79%, suggesting that the majority of jobs were completed without issues.

Total time statistic for 100 jobs (5 epochs):

Number of unsuccessful jobs: 21

Success rate: 79%

Description	**Total time**
Total time spent computing for successful jobs	13 days 6 hours 33 min 53 sec
Total time spent transferring files for successful jobs (last transfer, if there were retries)	1 day 21 hours 50 min 43 sec
Total time wasted due to transfers that failed (for successful jobs)	12 days 14 hours 12 min 50 sec
Number of jobs that failed (with no automated recovery)	18
Number of jobs that failed due to wrong output	3
Total time wasted in compute **due to preemption** (i.e. disconnects)	16 hours 1 min 29 sec
Total time wasted in file transfers for **preempted jobs** (last transfer, if there were retries)	47 min 35 sec
Total time wasted in compute **due to wrong outputs**	33 min 8 sec
Total time wasted in file transfer **for wrong outputs** (last transfer, if there were retries)	2 hours 11 min 8 sec
Total time wasted in compute **due ALL failed jobs**	16 hours 34 min 37 sec
Total time wasted in file transfer **for ALL failed jobs**(last transfer, if there were retries)	2 hours 58 min 43 sec
Mean time successful jobs took from first being matched to successful completion (starting file transfer to job termination)	8 hours 36 min 46 sec
Median time successful jobs took from first being matched to successful completion	4 hours 57 min 25 sec

Across the 100 jobs, the majority were run on three sites: CHTC-Spark-CE1, PDX-Coeus-CE1, and GP-ARGO-ksu-backfill. The jobs were divided into two queues of 50 jobs each.

Job Execution Performance Analysis

The success rate of the jobs was 79%, indicating that a significant majority of the jobs were completed successfully. For these successful jobs, the total time spent on computing was 13 days, 6 hours, 33 minutes, and 53 seconds. Additionally, file transfers for these jobs took 1 day, 21 hours, 50 minutes, and 43 seconds (the last transfer if there were retries). However, there was considerable time wasted due to failed file transfers, amounting to 12 days, 14 hours, 12 minutes, and 50 seconds, despite these jobs ultimately being successful.

In terms of job failures, 18 jobs did not recover automatically, and 3 of these failures were due to incorrect output. The time wasted in computing due to preemption—such as job disconnects—was 16 hours, 1 minute, and 29 seconds. File transfers for preempted jobs added an additional 47 minutes and 35 seconds of wasted time. Moreover, the total time wasted in computing due to wrong outputs was 33 minutes and 8 seconds, with file transfers for these wrong outputs contributing another 2 hours, 11 minutes, and 8 seconds. Overall, the total time wasted in computing for all failed jobs was 16 hours, 34 minutes, and 37 seconds, while the time wasted in file transfers amounted to 2 hours, 58 minutes, and 43 seconds.

Regarding performance metrics, the mean time from when a job was first matched to its successful completion, including file transfer and job termination, was 8 hours, 36 minutes, and 46 seconds. The median time for successful jobs was 4 hours, 57 minutes, and 25 seconds.

Overall, while most jobs succeed, the system loses considerable time due to failed transfers, preemptions, and incorrect outputs, indicating areas for potential improvement.

Final Conclusion:

Training the ResNet50 model on the ImageNet dataset across different methods on the Open Science Grid (OSG) provided valuable insights into dataset handling and job execution efficiency.

Method 1: Traditional Download and Extraction This method was unsuccessful due to HTCondor's 5 GB input file transfer limit. Despite efforts to split the dataset into smaller chunks, the total size of the input files consistently exceeded the limit, preventing successful execution.

Method 2: Direct Fetching from Origin with HTTP Method 2 also faced challenges. The initial file transfer issues led to job eviction due to lack of progress, likely caused by network or server problems. Attempts to use `wget` for downloading the dataset failed due to the unavailability of the command in the execution environment. Consequently, this method was deemed unsuccessful.

Method 3: Direct Fetching from Origin Using OSDF Protocol Method 3 proved successful, leveraging the Open Science Data Framework (OSDF) to efficiently manage the ImageNet dataset. After overcoming initial issues with file transfers and environment configurations, the ResNet50 model was trained for 20 epochs successfully. Testing across various sites demonstrated the robustness of this method. The 5 epoch runs showed that stalling during file transfers posed a challenge that, if mitigated, would significantly enhance overall efficiency.

Overall, Method 3 was the most effective approach for handling large datasets on OSG, highlighting the importance of efficient data transfer protocols and site-specific performance optimization. This method demonstrated the ability to handle large datasets effectively while ensuring robust model training across different environments. Addressing the stalling issues observed in file transfers would make this approach even more reliable and efficient for large-scale distributed training on the OSG.

Future Scope:

In summary, the key areas for improvement are optimizing file transfer processes to minimize inefficiencies, enhancing automated recovery and failure management to reduce the impact of job failures, and striving for more consistent job completion times. By focusing on these areas, the system's overall performance and reliability can be significantly improved.