IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.151104

Intelligent Backup Scheduling for Predictive Storage Management

Jatin Verma[1], Akash Kumar[2], Abhishek Jatla[3], Dr. P. Durgadevi[4]

Department of Computer Science and Engineering, SRM Institute of Science and Technology, Vadapalani Campus, Chennai, India[1-3]

Assistant Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, Vadapalani Campus, Chennai, India[4]

Abstract: Data is a central part of everyday digital work. Students, developers, companies, and researchers keep documents, source code, project files, reports, images, databases, and other valuable information on their computers. Such data can be lost through accidental deletion, hard-disk failure, software crashes, malware, power failure, or file corruption, which makes regular backups necessary. Many conventional backup systems follow a fixed schedule, such as a backup every day or every week. This approach is easy to use, but it ignores whether the files have actually changed. A backup may be created even when nothing important has changed, which wastes time, storage space, and system resources. Likewise, if a large amount of data changes right after a scheduled run, the new data stays unprotected until the next one. This project proposes an adaptive backup scheduling system developed in Python. The system monitors a selected source folder and compares its current files with a previous snapshot. New, modified, and deleted files are identified from the file path, file size, and last-modified time. A backup is triggered only when the amount of changed data reaches a configured threshold or when too much time has passed since the last successful backup. CPU usage and free space are also checked before a backup begins. The current prototype uses incremental backup, so only new or modified files are copied to the destination. Each backup is saved in a separate timestamped folder, and a retention policy removes older folders beyond a fixed limit. The project also includes modules for logging, feature generation, training-data generation, and predictive storage analysis, which form a base for future machine-learning integration. The proposed system aims to make backups more efficient, reliable, and responsive to real changes in data.

Keywords: Adaptive backup, incremental backup, Python, file snapshot, storage monitoring, backup scheduling, predictive storage management.

I. INTRODUCTION

Digital data has become essential in education, business, software development, research, and personal computing. Students keep assignments, project reports, source code, datasets, and presentations on their laptops. Developers store application files, databases, configuration files, and code repositories. In all of these cases, losing important files can cause serious problems.

Data loss has many causes. A folder may be deleted by accident, an important file overwritten, the wrong drive formatted, or the Recycle Bin emptied too soon. Hardware problems such as hard-disk damage, SSD failure, power issues, overheating, or physical damage can also make files inaccessible, while software crashes, malware, ransomware, incomplete updates, and file-system corruption add further risk [1]. Keeping a single copy of important data is therefore unsafe.

Backup systems maintain another copy of the original data so that files can be restored if the original is lost or corrupted [2]. However, many simple systems rely on fixed schedules, for example a backup every day at midnight or every Sunday. The approach is simple, but it takes no account of how much data has actually changed.

A fixed schedule can produce unnecessary backups when only a small file has changed. It also creates exposure when a large amount of data changes after the previous run, because the latest version of that data remains unprotected until the next backup time. Backup systems should be able to examine current changes and decide based on real conditions.

Storage capacity is another important issue. If the backup drive becomes full, the backup may fail. Capacity forecasting in backup environments can help predict future storage requirements and warn administrators before the limit is reached [3], [4]. A reliable backup process should also verify that copied files are available and should be tested regularly for recovery [5].

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026
DOI: 10.17148/IJARCCE.2026.151104

This project proposes an adaptive backup scheduling system that uses file changes, elapsed time, CPU usage, and destination free space to decide whether a backup should be performed. The current version is rule-based, but it is designed so that machine-learning models can later be added to predict storage growth, backup size, and future capacity requirements.

II. PROBLEM STATEMENT

Traditional backup systems normally run at fixed times. Although this method is easy to configure, it may not suit different types of data and changing workloads. Backups may run even when no meaningful data has changed, and important file changes may remain unprotected until the next scheduled run. A backup may also start while the computer is already busy, or fail because the destination storage has too little free space. Old backup copies keep occupying storage if they are never removed, and the backup drive may be filling up without the user noticing until a failure occurs.

Another weakness is that a backup may be incomplete if files are locked, removed during copying, or affected by an error. In such a case, data may appear to be protected even though the backup process did not finish successfully. The project addresses the following question: how can a backup system automatically decide when a backup is necessary by observing file changes, system conditions, storage availability, and the time since the previous backup?

III. OBJECTIVES

The objectives of this project are:

1. To develop a Python-based backup system for protecting files in a selected folder.
2. To identify new, modified, and deleted files using folder snapshots.
3. To support incremental backup so that only changed files are copied.
4. To avoid unnecessary backups when there are no important changes.
5. To trigger a backup when the changed data size reaches a defined threshold.
6. To force a backup when too much time has passed since the last successful backup.
7. To check CPU usage before starting a backup operation.
8. To check free space in the backup destination before copying files.
9. To store backup copies in separate timestamped folders.
10. To keep only a configured number of recent backups through a retention policy.
11. To record backup details in logs for debugging and analysis.
12. To prepare the system for future machine-learning-based storage prediction.

IV. PROPOSED SYSTEM

This project implements an adaptive backup application in Python. Rather than creating a backup every time it runs, the application first checks whether the files in the selected folder have changed enough to need protection. It also checks the current system load, the space available in the backup drive, and the time elapsed since the previous backup.

The folder selected for protection in this implementation is C:\Users\SJ AKASHKUMAR\Desktop\ProjectData. The backup copies are stored separately at D:\Backup\ProjectData_Backup. The original files remain in the ProjectData folder on the Desktop, and when a backup is required, the relevant files are copied to the backup folder on the D: drive. Keeping the destination separate from the source makes backup copies easier to maintain and recover [6].

The application is designed to start automatically through Windows Task Scheduler, which can run the checking script every 15 or 30 minutes. The program does not create a backup at every check. Each scheduled run only examines the current condition of the source folder, and a backup is created only when the decision rules judge it necessary. The decision rests on four main factors:

The decision rests on the amount of data that has changed, the current CPU usage, the free space available in the backup drive, and the time since the last successful backup. This makes the backup process respond to actual file activity instead of relying only on a fixed backup time.

The working process can be summarized as follows. The system scans the source folder, creates a current snapshot of the files, and compares it with the previous one to identify new, modified, and deleted files. It then calculates the amount of changed data and checks CPU usage, free storage, and backup age before deciding whether to back up, skip, or delay the operation. If a backup is required, the changed files are copied, after which the backup is verified and the log and snapshot are updated.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.151104

A. Configuration of the System

The main settings are stored in a file named config.py. Keeping these values in one place makes the system easier to modify. The source folder, backup location, or decision thresholds can be changed without touching the backup code itself.

The system uses incremental mode by default. If only one file has changed, the application does not copy the entire project folder again; it copies the changed file and records the operation in the backup log. If no meaningful change is detected and the last backup was recent, the operation is skipped. This configuration-based design also keeps the project flexible: thresholds can be adjusted for different users or storage environments, and the rule-based decision process can later be replaced or extended with a machine-learning model.

TABLE I. MAIN CONFIGURATION SETTINGS

Setting	Current value	Purpose
Source folder	C:\Users\SJ AKASHKUMAR\Desktop\ProjectData	Folder being protected
Backup folder	D:\Backup\ProjectData_Backup	Location for backup copies
Backup mode	Incremental	Copies only new or modified files
Minimum changed data	5 MB	Triggers a backup when this amount changes
Maximum CPU usage	60%	Delays non-urgent backup activity when the system is busy
Forced backup interval	24 hours	Ensures that a backup occurs at least within this period
Minimum free space	500 MB	Prevents a backup when the destination has very little space
Maximum backups retained	5	Limits the number of stored backup versions

V. SYSTEM MODULES

config.py

The config.py file contains the main settings of the backup system. It stores source and destination paths, backup mode, thresholds, storage limits, and state file locations. The folder to protect or the backup location can be changed by modifying the settings in this file.

snapshot.py

The snapshot.py module creates a record of the files currently present in the source folder. For every file, the snapshot stores its relative path, file size, and last-modified time. The previous snapshot is compared with the current one to identify new, modified, and deleted files, which allows incremental backup instead of copying every file every time.

backup.py

The backup.py module performs the actual file copying. Before copying, it checks whether enough free space is available on the destination drive. Two backup modes are supported: incremental backup, which copies only new and modified files, and full backup, which copies all files from the source folder. Each backup run creates a timestamped folder, for example D:\Backup\ProjectData_Backup\2026-08-08_16-40-00. A manifest file named _manifest.txt is created inside each backup folder, recording the backup mode, the number of copied files, and information about deleted source files.

decision.py

The decision.py module contains the adaptive decision logic. It checks the amount of changed data, the time since the previous backup, CPU usage, and the threshold values from config.py. The following rules can be applied.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.151104

A backup is triggered when changed data is 5 MB or more, and also when 24 hours have passed since the last backup, even if the changes are small. If CPU usage is above 60%, a non-urgent backup is skipped or delayed, and if the backup drive does not have enough free space, the backup is skipped.

check_and_backup.py

The check_and_backup.py file connects the main modules together. When it runs, it loads the source-folder setting and the previous snapshot, creates a new snapshot, finds changed and deleted files, and calls the decision module. The backup runs only if required, after which the updated snapshot and the latest backup time are saved. This file can be configured in Windows Task Scheduler.

logger.py

The logger.py module records backup-related information such as date and time, number of files changed, amount of changed data, backup decision, backup status, number of files copied, backup destination folder, and errors. These logs help with debugging and can later serve as historical data for machine-learning training.

features.py

The features.py module is designed for feature engineering. It converts raw storage and backup logs into values a machine-learning model can use. Possible features include storage used, free storage, changed-data size, number of changed files, time since the previous backup, backup duration, CPU usage, day of the week, and hour of the day.

generate_training_data.py

The generate_training_data.py file prepares historical data for model training. It can read backup logs and storage records, create features, and save a structured dataset such as training_data.csv. The target may be future storage usage, expected backup size, or the possibility of a storage shortage.

model.py

The model.py module holds the prediction logic. At present, the project uses rule-based logic. In future, this file can contain a trained machine-learning model that predicts future storage usage, expected backup size, expected backup duration, storage-capacity risk, and the time remaining before storage reaches a critical limit.

train_model.py

The train_model.py script trains the machine-learning model. It loads the training dataset, selects features and target values, splits the data into training and testing sets, trains the model, evaluates its accuracy, and saves the trained model as a file such as storage_model.pkl.

A. State Folder

The state folder stores information needed between executions. It can contain last_snapshot.json, last_backup_time.txt, and backup_log.csv. The last snapshot stores the previous source-folder state. The last-backup-time file records the most recent successful backup, while the CSV file stores backup activity and decisions.

VI. ADAPTIVE BACKUP DECISION PROCESS

The adaptive backup process is carried out in the following steps:

1. The checking program is started automatically at regular intervals using Windows Task Scheduler.
2. The previous snapshot is loaded from the state folder.
3. A current snapshot of the source folder is created.
4. The old and new snapshots are compared.
5. The total size of new and modified files is calculated.
6. The system checks whether changed data is at least 5 MB, whether 24 hours have passed since the last backup, whether CPU usage is below 60%, and whether sufficient free space is available in the destination drive.
7. The system decides whether to trigger, skip, or delay the backup.
8. If a backup is triggered, changed files are copied to a timestamped destination folder.
9. The copy is verified and a manifest is created.
10. The latest snapshot, backup time, and log are updated.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.151104

VII. PREDICTIVE STORAGE MANAGEMENT USING MACHINE LEARNING

The current project is mainly a rule-based adaptive backup prototype, but it also includes the structure needed to add machine learning. Historical data can be used to predict future storage requirements. The model can draw on daily storage usage, daily data growth, number of changed files, backup size, backup duration, CPU usage, time features, previous backup failures, and available free storage.

The model may predict storage usage after 7 or 30 days, expected backup size, expected backup duration, probability of storage shortage, and time remaining before reaching 80% or 90% capacity. Forecasting of this kind is well established in backup environments [3], [4], and a simple forecast can be represented as:

$$\textit{Predicted Storage Usage} = \textit{Current Storage Usage} + (\textit{Average Daily Growth} \times \textit{Number of Days})$$

For example, if current usage is 60 GB and average daily growth is 1 GB, the estimated usage after 7 days is 67 GB. If total capacity is 80 GB, the system can alert the user before storage becomes critical. Possible actions based on the prediction include notifying the user, recommending extra storage, moving old backups to external or cloud storage, adjusting retention settings carefully, changing backup frequency, or scheduling large backups during low-load periods.

VIII. TESTING METHODOLOGY

The project can be tested using the following scenarios.

Test Case 1: First Backup. Files are created in the source folder and the system is run for the first time. Since no previous snapshot exists, all files are treated as new and a backup should be created on the D: drive.

Test Case 2: No File Changes. The system is run again without changing any file. It should detect no significant change and skip the backup because the previous backup is recent.

Test Case 3: Small File Change. A small text file is edited and the system is run. If the change is below 5 MB and the previous backup is recent, the backup should be skipped.

Test Case 4: Large Data Change. A file larger than 5 MB is added or modified and the system is run. The change should be detected as above the threshold, and an incremental backup should be created.

Test Case 5: Forced Backup After 24 Hours. If no major changes occur but 24 hours have passed since the previous successful backup, the system should still trigger a backup.

Test Case 6: Low Storage Space. If the destination drive has less than 500 MB of free space, the backup should not start and the system should record a warning.

Test Case 7: Retention Policy. More than five backup runs are created. The system should remove the oldest backup folders and retain only the five most recent ones.

IX. EXPECTED RESULTS

The proposed system is expected to make a backup decision each time the scheduler runs, rather than creating a backup automatically on every execution. The decision is based on file changes, elapsed time, CPU usage, and available storage space.

TABLE II. SCENARIOS AND EXPECTED SYSTEM ACTIONS

Scenario	Expected system action
First run	The system triggers an initial backup because no previous backup exists.
No changes after a recent backup	The system skips the backup and records the decision in the log.
Small change below the configured threshold	The system skips the backup if the change is not considered significant.
Large change above the configured threshold	The system triggers an incremental backup containing the new or modified files.
More than 24 hours since the last backup	The system triggers a backup regardless of the amount of data changed.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.151104

CPU usage above the configured threshold	The system delays or skips a non-urgent backup to reduce system load.
Insufficient free space on the destination drive	The system skips the backup and records a warning.
More than five backup folders exist	The system removes the oldest backup folder according to the retention policy.

The incremental approach is expected to reduce unnecessary storage use and copying time, since only new and modified files are transferred. Timestamped folders make backup versions easier to identify and restore, and the log file provides a record of backup decisions, system conditions, successful operations, skipped backups, and errors.

The expected behavior will be verified through test cases covering initial backup creation, unchanged files, small and large file modifications, high CPU usage, low destination-space conditions, and backup-retention limits. Actual test results should be added to the report once testing is complete.

X. ADVANTAGES OF THE PROPOSED SYSTEM

The system avoids blindly creating a backup at every scheduled check and instead uses actual file changes to decide when one is needed. Incremental backup reduces copy time and storage use, CPU load is checked before a backup starts, and free storage space is checked before copying begins. A backup is also forced after a defined time period for safety.

Separate timestamped backup folders and retention management keep a usable backup history, and backup details are recorded in logs. Because the system is implemented in Python, it is easy to understand and modify. It can be integrated with Windows Task Scheduler for automatic execution and extended with machine learning for storage forecasting.

XI. LIMITATIONS

The current prototype has some limitations. It protects only the configured source folder, and the backup destination is a local D: drive, which may not protect against theft, fire, or complete system failure. A local backup on the same computer may also be affected by ransomware or hardware failure [1].

The decision system is rule-based and not yet fully ML-driven, and a machine-learning model needs sufficient historical data before it can make reliable predictions. CPU percentage alone may not represent all system-load conditions, and file size and modification time may not detect every possible type of internal content change. Automatic storage expansion requires integration with cloud or storage APIs, while automatic storage shrinking should be used carefully because deleting or reducing backup capacity can affect recovery.

XII. FUTURE SCOPE

The system can be improved in the following ways:

- Integrate Windows Task Scheduler for fully automatic periodic checking.
- Add a dashboard for backup history, storage usage, and prediction results.
- Connect with cloud storage platforms such as Google Drive, AWS S3, Azure Blob Storage, or OneDrive.
- Add encryption for sensitive backup data.
- Use file hashes to verify that copied files are identical to the originals.
- Add email or notification alerts for backup failures and low storage.
- Use machine-learning models to predict future storage usage.
- Support multiple source folders and multiple users.
- Add database backup support.
- Add automated recovery testing [5], [6].
- Use reinforcement learning to improve backup scheduling decisions over time.
- Add ransomware detection and immutable backup copies [1].
- Support external drives, NAS devices, and distributed storage environments.

XIII. CONCLUSION

This paper presented an adaptive backup scheduling system developed in Python. The system protects a selected source folder by monitoring changes in files and copying data to a separate backup location. Instead of performing a backup



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026
DOI: 10.17148/IJARCCE.2026.151104

every time the program starts, it checks the amount of changed data, CPU usage, available storage space, and the time since the previous backup.

Snapshots are used to identify new, modified, and deleted files, and incremental backup ensures that only changed files are copied. Backup copies are stored in timestamped folders on the D: drive, which makes them easier to manage and restore, and a retention policy limits the number of stored versions.

The current implementation is a rule-based adaptive backup system. It already includes modules for logging, feature generation, training-data generation, and machine-learning integration. In future, historical logs can be used to predict storage growth and warn users before the backup destination reaches full capacity.

Overall, the project offers a useful and practical approach to improving backup reliability. It reduces unnecessary backup operations, improves storage management, and lays a foundation for future intelligent backup and predictive storage systems.

REFERENCES

[1] National Institute of Standards and Technology, “Protecting Data from Ransomware and Other Data Loss Events.”
[2] United States Environmental Protection Agency, “Protect: System Backups.”
[3] M. Chamness et al., “Capacity Forecasting in a Backup Storage Environment,” USENIX LISA.
[4] “Soothsayer: Predicting Capacity Usage in Backup Storage Systems,” ACM/IEEE MASCOTS.
[5] National Institute of Standards and Technology, “Security Guidelines for Storage Infrastructure,” NIST Special Publication 800-209.
[6] National Institute of Standards and Technology, “OT Backup Quick Start Guide,” NIST Special Publication 1339.