IJMSERH

International Journal of Multidisciplinary and Scientific Emerging ResearcH (IJMSERH)

Volume 13, Issue 2, April-June 2025

Impact Factor: 9.274

International Journal of Multidisciplinary and Scientific Emerging ResearcH (IJMSERH)

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302080

Transforming Static Server Allocation into an Adaptive Compute for Enhanced Throughput and SLA Compliance

Nagabhushanam Bheemisetty

Senior Architect, Capgemini Financial Services USA Inc., USA

ABSTRACT: The research suggests a system of managing compute resource hardware management called Priority-Based Optimized Compute Resource Management to improve batch processing efficiency in a computing context. It manages compute resources across a series of data centers and disaster recovery sites, combining an adaptive microservices layer and both AutoSys and Univa Grid Engine schedulers. It also has a method of task and job prioritization based on multi-factor algorithms assessing workloads on the cluster, criticality of applications and client implications. While AutoSys prioritizes the jobs, the microservices layer optimizes resources dynamically. Univa Grid Engine optimizes scheduling based on server conditions. The framework commits to data analytics and machine learning for monitoring and in-flight modification of it resource management being the biggest factor in manage workload when accomplish the outcomes of increase compute efficiency (35-40% improvement), I.T infrastructure utilization (25% improved), reduced SLA violations all without adding any additional hardware. As a methodology it takes the static enterprise batch system and turns it into a structured, dynamic self-optimization system to allow for better workload orchestration and ability to adapt in cloud environments.

KEYWORDS: Priority-Based Optimized, Batch Processing, Disaster Recovery, SLA Compliance

I. INTRODUCTION

Ecosystems for batch processing utilize coordinated scheduling and execution technologies like AutoSys and Univa Grid Engine to automate data operations with minimal human intervention. Jobs are submitted programmatically through APIs and follow a structured two-layer scheduling process, where AutoSys handles initial queuing and priority scoring, while Grid Engine allocates jobs to optimal resources based on specific requirements. Essential activities in these ecosystems include reliable data transport, transformation, and reporting, which are crucial for both cloud-native and hybrid environments. However, batch processing often faces challenges such as data latency and gaps, duplication and inconsistency, schema incompatibility, dependency management issues, resource bottlenecks, and the need for effective error handling and recovery. To address these challenges, an optimized framework incorporates exact-once processing assurances, dynamic task prioritization, and continuous monitoring, which collectively enhance throughput, reduce data duplication, and improve fault tolerance. Additionally, systematic server snapshots and adaptive resource allocation facilitate quick recovery and automated corrections during failures or performance issues.

Batch pipeline failures may arise from a variety of causes such as modifications to schemas, upstream availability of data, insufficient resources, issues in the code, broken dependencies, data quality issues, inadequate error handling and lack of retries, inadequate monitoring and alerts, configuration drift, and unexpected surges in data volume. Schema modifications can cause format errors or lead to expected fields being absent or altered leading downstream batch jobs to fail. Upstream data availability can cause pipeline failure, and lack of resources can cause batch jobs to crash or perform poorly. Bugs in the code can result in unanticipated jobs crashing or providing unexpected results. Dependency failures can propagate to cascade failures. Data quality issues can arise from poor error handling, resulting in either corrupt input data resulting with processing errors, or silent data corruption. Bad error handling and lack of retries can result in the production of duplicate data, or pipelines finishing in inconsistent states. Insufficient found monitoring and alerting can result in quiet failures, outages of pipelines, or long periods of undetected data corruption. Progressive configuration drifts can occur when credentials, environment variables or service configurations gradually change. Surges in data volume can cause slow down or crashes. Bringing these issues to a resolution requires comprehensive schema management, error-tolerant retries, resource scaling, thorough testing, and broad Observability [2]. Batch processing environments are likely to experience inadequate use of computing power in multiple fulfillment applications, leading to resource contention, tardy processing, and high costs.

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302080

It is due to tightly allocated or inadequately planned computational resources like CPU, memory, and I/O bandwidth. It causes inefficiencies due to unbalanced work scheduling, inadequate dynamic scaling or prioritization schemes, and inadequate coordinated resource management. To mitigate this issue, there must be a best workload orchestration and resource management solution. This approach supports efficient, equitable, and dynamic distribution of computation resources across batch jobs of multiple fulfillment programs. Centralized scheduling and orchestration minimize contention and idleness, while cloud auto-scaling or container orchestration support dynamic scaling. Resource quotas and priorities allow balanced use and prevent monopolization by a few jobs. End-to-end monitoring and autoscaling provide optimal utilization and SLA conformance. Job resource profiling helps minimize over-provisioning and issue more precise demands for resources. Load balancing and throttling algorithms can assist in limiting bottlenecks by spreading workload across accessible resources in a priority-based manner. By removing idle and misused compute times through better orchestration, dynamic scaling, and monitoring, batch processing environments can achieve higher productivity, reduced costs, and higher throughput. Priority-Based Optimized Compute Resource Management Framework is a framework that assigns compute resources based on the priority of incoming jobs or tasks. It ensures more efficient allocation of higher-priority jobs, even if it means shifting resources away from lower-priority jobs. This process reduces waiting for important activities, increases system efficiency, and maximizes resource usage while still ensuring work usefulness. Benefits of this model include effective use of resources, low job latency, increased fairness, improved scalability, and prevention of starvation. To tackle inefficient use of the compute in batches, the framework can be utilized by assigning priorities to fulfillment jobs based on their operation urgency, SLA priority, or business impact. A priority-based scheduler can be added centrally to assess the availability of resources and accept the requests with priority information [3]. Dynamic resource allocation is achieved by giving preference to compute resources towards higher-priority batch jobs and offloading resources from lower-priority or idle processes as required. Resource monitoring facilitates dynamic reallocation for maximum use so that lower-priority work gets timely access to resources without data inconsistency or cascade failure.

Adjustments can be made through feedback and optimization on resource plans and priorities for upcoming batch cycles. Through optimized task completion and real-time dynamic resource allocation, the Priority-Based Optimized Compute Resource Management Framework provides an orderly, dynamic compute waste solution with enhanced fulfillment performance, processing time reduction, and resource usage with defined business goals as a focus [4]. Multiple inputs from business are used to determine dynamic scheduling parameters like job priority, urgency score, cluster weight, and scheduler priority through a priority-based multi-factor weighted approach. This approach utilizes math models with concepts borrowed from batch orchestration engines (AutoSys, Airflow, Control-M) and CPU scheduling (Priority, Shortest Job First, Multilevel Queue).

The algorithm calculates the overall Scheduling Priority Score (SPS) for each batch job from weighted inputs of technical and business parameters. The algorithm uses declining SPS values for assigning workloads to compute nodes, with high SPS indicating greater priority for scheduling. Longer waiting times or SLA violations are given higher urgency weight dynamically. Cluster weight is computed using the Maximum Capacity Current Load (CW) to reroute new jobs to lightly loaded clusters and penalize saturated clusters. The algorithm synchronizes scheduling priorities with AutoSys, which supplies scheduling priority levels dynamically based on computed SPS values. This leads to balanced scheduling, end-to-end optimization, priority adjustments for real-time scheduling, fairness among business-critical and infrastructure-bound workloads, and resource optimization.

To implement this algorithm, use a microservice or Fabric/DataStage layer that computes SPS per scheduling cycle, place computed scores into the batch orchestrator's metadata layer, enable dynamic priority scheduling by script triggers or the orchestrator's API, and rebalance weights from time to time using real resource consumption data and past SLA completion. This algorithmic approach converts unprocessed business and operational data into measurable weights that are plugged into scheduling logic, giving important jobs proper scheduling priority and offering a data-driven, flexible, and scalable way to calculate optimization [5].

II. RELATED WORK

This summary explains several methods for scheduling chemical batch processing depending on some priority rules. Trautmann et.al., [6] presents priority-rule heuristics and prioritizes tasks according to job criticality and precedence constraints, arriving at the conclusion that compound priority rules combining processing time windows equated to business priorities produce optimal throughput in complex batch systems. The results highlight some of the basic algorithms- Round Robin (RR), Priority Scheduling, Shortest Job First (SJF), and First Come First Serve (FCFS) and how effect waiting time, risk of starvation, and latency. Full C++ and Java code for Priority Scheduling is also included

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302080

- the solution of aging to avoid starvation, and the meaningful approach of genuine time work turnaround and wait time calculations.

Madhusmita et.al., [7] presents a new multicore systems priority-based task scheduling algorithm with the focus on load balancing across more than one core processor. This approach ensures high-priority tasks are finished before core overload, enhancing system responsiveness and energy efficiency in low-computing contexts. Additionally, various studies also focused on heuristic scheduling algorithms such as rule-based heuristics, heuristic job sequencing, and Critical Path-Based Optimization. Key insights from these studies are the efficacy of priority scheduling based on rules for heterogeneous batch workloads, the need for priority aging and preemption in modern-day compute management, and the adaptability of multi-factor scheduling in hybrid systems.

From research on scheduling chemical batch processes with priority rules, operating system CPU scheduling, priority CPU scheduling program, a novel algorithm for multicore systems' priority-based task scheduling, and industrial applications research on rule-based and heuristic scheduling. The first study proposed priority-rule heuristics for scheduling chemical batch processes, giving tasks dynamic weights in terms of resource contention and urgency. The second study organized theoretical and practical explanations of basic scheduling algorithms like FCFS, SJF, Priority, and Round Robin and brought about the idea of aging to prevent infinite waiting for low-priority operations. The third study introduced implementation-level models to model preemption-based priority-based scheduling, which formed the foundation for modern-day task orchestration logic. The fourth research developed a hybrid priority and load-balancing model for multi-core processors, adaptively adjusting time slices and processor affinity according to task priority. The fifth research explored metaheuristic and heuristics scheduling algorithms for industrial batch processes, paving the way for modern-day Priority-Based Optimized Compute Resource Management Frameworks. The sixth study provided mathematical formulas for batch scheduling constraint-based optimization and mixed-integer programming, impacting research on dynamic resource allocation for multi-application clusters. The seventh study investigated multi-objective scheduling for batch execution on the cloud, highlighting the need to utilize priority-based adaptive models to dynamically adjust compute allocations.

AutoSys has employed several techniques to enhance its workload orchestration system. Multi-factor weighted priority calculation is one of them, in which a weighted composite priority score is assigned to each job based on a combination of business and operational parameters. The formula employs queue delay analysis and historical SLA information to dynamically change weights, setting queue precedence in numerical form. SLA aging models are another technique used for estimating urgency scores that grow nonlinearly as deadlines are near. This ensures near-real-time response to unmet commitments via dynamic recalibration throughout each scheduling cycle. Cluster weight and dispatching aware of load are also used in AutoSys "factor" and "load" properties. Each cluster or machine is given a modifiable "factor weight" to indicate processing capacity, allowing schedulers to prefer machines with larger free capacity and lower active load weight when launching new jobs. Another method is box-level and hierarchical priority assessment, where sub-jobs are ranked according to job-level priorities and internal dependence criteria inside each box. This allows high-priority parent boxes to temporarily reclassify child operations to avoid bottlenecks in important pathways [8].

Dynamic queue rebalancing and custom reference tables are also utilized in Stack Overflow AutoSys designs and implementation studies. These techniques record unprocessed feeds and corresponding business priorities in reference tables that developers keep up-to-date. Dynamic queue rebalancing and custom reference tables provide fine-grained control over reordering batch pipelines without completely rearranging AutoSys dependencies. Queue priority optimization vs. job load is another method used in cloud-ready extensions and AutoSys 11.x+. This method considers both job_load and priority when choosing jobs at runtime, ensuring that important but light jobs are preempted over heavy or non-urgent ones. Lastly, iterative learning and feedback-based adaptation are used in subsequent expansions of machine learning and heuristics, leading to the development of semi-autonomous optimization models, which were precursors to AI-driven workload orchestration found in intelligent corporate schedulers today [8].

The methodology for assessing the effectiveness of a composite priority scoring system in scheduling. It begins with a historical correlation of results, which is used to determine the correlation between allocated priority scores and past task results. This is based on history-based prioritizing techniques that used past response accuracies and defect rates to justify test case selection algorithms. The next step is testing for predictive accuracy using regression or classification models. A prediction model is trained using historical batch task data, and individual priority characteristics (CP, OP, and AP) are used as input features. Metrics such as accuracy, F1 score, or area under the ROC curve (AUC) are employed for comparison of actual and predicted high-impact outcomes. The CPS weights are tuned so that the accuracy of the model becomes constant [9].

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302080

It concludes and proceeds to rank order validation through an analytical hierarchy process. The ranking order is tested against human or rule-based priority orders based on order reversal metrics. The new composite scoring technique can substitute for legacy prioritization in the event of less ranking reversals with the preservation of the original priority justification. It proceeds to component and composite outcome analysis, with multi-component outcome analysis employed in validating the CPS. The model is taken to be a composite variable that forecasts multifactorial performance, including throughput, cost efficiency, and SLA compliance. Time-to-event analysis is then used to confirm whether occupations with higher CPS have lower risk of breach. It then ends with backtesting or historical simulation, where the new composite score is used to replay prior scheduling cycles and compare outcomes to original ones. The priority scoring model is validated by a positive consistent score. Lastly, the synopsis ends with ongoing validation through feedback loops by means of rolling window analysis [10].

Composite priority score is a metric employed to quantify business effect and results of operations. It is built on customer, order, and application priorities and is graded with different metrics. Some metrics used to evaluate it are Spearman's Rank Correlation (ρ), Normalized Discounted Cumulative Gain (NDCG), Kendall's Tau, Predictive Accuracy, Impact and Efficiency Measures, Business Effectiveness, Fairness and Stability, and a Holistic Method for Verification. Ranking metrics evaluate how well the computed priority ratings reflect the real or desired order of task importance. Business Effectiveness measures equate real-world outcomes with technical metrics. Spearman's Rank Correlation (ρ) calculates the relationship between actual observed ranks and expected job rankings. The Normalized Discounted Cumulative Gain (NDCG) assigns greater weight to accuracy toward the top of the list to assess ranking quality. Kendall's Tau evaluates the degree of rank order agreement between manually stated priorities and those generated by the model. The Priority-Based Optimized Compute Resource Management Framework method differs from traditional static resource allocation in several aspects. It involves rigid pooling of resources based on workload or application, a single computing grid for primary and disaster recovery settings, adaptive load balancing, dynamic scheduling using composite scores, managing priorities, recovery from disasters, compliance with SLAs, scalability, handling failure and recovery, and cost of operations. It also includes a mechanism for scheduling static schedules using composite scores for dynamic, priority-driven real-time scheduling, and simple, remote monitoring through constant telemetry and feedback loops for capacity planning and tweaking. The framework also addresses issues with failure propagation and kill signals, and the need for physical intervention for scaling. By incorporating these improvements, the Priority-Based Optimized Compute Resource Management Framework can lead to better operational efficiency and lower costs is shown in below Table 1 [11]:

Aspect	Traditional Static Resource Allocation	Priority-Based Optimized Compute Resource Management Framework
Resource Pooling	Rigid, siloed per application or workload	Unified compute grid across primary and DR environments
Resource Utilization	Inefficient, with over/under-utilized servers	Dynamic load balancing maximizing accepted resource usage
Scheduling Mechanism	Static schedules, fixed allocation	Dynamic, priority-driven real-time scheduling using composite scores
Priority Handling	Limited or none, often FIFO or static priorities	Composite priority including customer, order, application, urgency
Disaster Recovery Usage	Idle, used only during failover	Actively used for non-critical workloads, increasing throughput
SLA Compliance	Frequent misses under peak loads due to rigid allocation	Improved SLA adherence via priority-aware resource redistribution
Scalability	Difficult scaling, manual intervention required	Adaptive scaling with workload-aware adjustments
Failure & Recovery Handling	Limited failure propagation awareness	Custom failure propagation and kill signal handling
Operational Cost	Higher due to static overprovisioning	Lower – better return on investment through infrastructure reuse
Observability & Feedback	Basic, isolated monitoring	Continuous telemetry and feedback loop for tuning and capacity planning

Table 1: Comparison of **Prior Traditional Static Resource Allocation** versus the **Priority-Based Optimized Compute Resource Management Framework**

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302080

Predictive Accuracy measures assess how effectively the score forecasts actual results, including revenue preservation, reduced SLA violations, or timely work completion. Precision@K and Recall@K measure the degree to which the top K prioritized jobs match the real high-impact jobs. The F1-Score balances recall and accuracy to account for the trade-off between missing important tasks and overprioritization. Impact and Efficiency Measures confirm if improved scores result in noticeable operational improvements. TIR stands for throughput improvement ratio, while SLA Adherence Rate and Compute Utilization Efficiency evaluate the increase in jobs done per unit of time following the implementation of priority scoring. GAP Avoiding Costs Savings determines how much money is avoided being spent by more important jobs, avoiding fines or lost revenues. The Customer Satisfaction Index (CSI) contrasts the distribution of priorities among work with customer KPIs after the implementation. The Priority Accuracy Index (PAI) guarantees that the score behaves consistently and impartially across client tiers or apps. An exhaustive approach to verification includes a combination of ranking metrics (Spearman's ρ, NDCG), employing predictive metrics (AUC, F1, MAE), and tracking business KPIs (cost-effectiveness, TIR, and SLA compliance) to prove results [12].

III. ARCHITECTURE

The system for job submission utilizes a specific JSON format known as REST API JSON and supports various computing requirements through a software environment that includes Java 17, C/C++, Python, and Bash Shell. It can handle different types of supporting file systems, such as bfs, efs, and cfs, and has a capacity of 32 slots for concurrent job execution. The scheduling architecture is structured in two levels. The first level, managed by AutoSys, serves as the main entry point for job submissions, where jobs are queued and assigned initial priority scores based on various factors. These scores dictate the order of job dispatching. The second level involves the Univa Grid Engine, which is responsible for executing jobs on computing nodes. It manages resources through its internal queues and assigns tasks based on available computing resources, required software platforms, necessary file system mounts, and job classifications. This two-level scheduling system ensures efficient resource utilization and optimal job execution. Additionally, the system calculates two sets of priorities: one for AutoSys to determine the initial job queue and another for the Grid Engine to refine execution order based on resource constraints and compatibility.

The process of managing job dispatch in a compute cluster involves several key steps. Initially, a real-time snapshot of the cluster's compute nodes is obtained from the Grid Engine's master node, which includes crucial information such as the number of available slots, CPU and memory usage, node status, and the availability of necessary file systems and software environments. This snapshot provides a comprehensive view of the current resource environment. Next, a composite dispatch score is calculated for each pending job, taking into account various factors like application priority, customer needs, job specifics, cluster conditions, and wait times. This score is essential for determining how tasks will be assigned based on the live resource snapshot.

Following this, jobs are assigned to appropriate compute nodes based on their dispatch scores and the availability of resources. The assignment process considers the capacity of candidate servers, the availability of required software and file systems, job classification alignment, and overall cluster load balancing to optimize resource usage. Once suitable nodes or queues are identified, jobs are dispatched to the selected servers for execution. To prevent resource contention, the scheduling system locks slots on these servers, ensuring that the necessary resources are reserved for the dispatched jobs. Finally, the system continuously updates its knowledge of available resources by frequently querying the Grid master. This allows for dynamic adjustments to the dispatch logic in response to changes in the cluster, such as nodes becoming busy or new nodes being added.

The scheduler queues assign a dispatch score to each pending work based on various priority parameters, including application priority, customer priority, inherent importance, and cluster priority. These factors are weighted to create a composite Dispatch Score, which can be adjusted as needed. To avoid starvation, the algorithm also incorporates Wait-Time precedence, ensuring that tasks that have been in the queue longer receive higher priority over time. After calculating the dispatch scores, tasks are evaluated and dispatched to compute nodes based on their scores and available resources. The system includes features for continuous monitoring of job completion, dynamic recalibration of priorities, and workload redistribution to optimize resource utilization, including transferring workloads to idle or disaster recovery servers.

The algorithm for managing jobs in scheduler queues calculates a dispatch score based on several priority parameters, including application priority, customer priority, job priority, and cluster priority. Each job's score is a composite of these variables, which can be adjusted through weights. To prevent starvation and ensure that longer-waiting tasks are prioritized, the algorithm also incorporates a Wait-Time precedence. Once dispatch scores are established, jobs are

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302080

prioritized and assigned to compute nodes based on their ratings and available resources. The system continuously recalibrates priorities and redistributes workloads as needed, including reallocating tasks to idle or disaster recovery servers to optimize resource utilization while tracking job completion is shown in below Figure 1:

```
# Configurable weights for priority factors
WEIGHT_APP_PRIORITY = 0.3
WEIGHT_CUST_PRIORITY = 0.25
WEIGHT_JOB_PRIORITY = 0.3
WEIGHT_CLUSTER_PRIORITY = 0.15
WEIGHT_WAIT_TIME = 0.2  # Wait-time priority factor
# Function to normalize scores to [0,1]
def normalize(score, min_val, max_val):
    return (score - min_val) / (max_val - min_val) if max_val > min_val else 0
# Calculate composite priority score for a job
def calculate_dispatch_score(job, current_time):
    # Extract raw priority values from job metadata
    app_priority = job.application_priority
    cust_priority = job.customer_priority
    job_priority = job.intrinsic_job_priority
    cluster_priority = job.cluster_priority
    # Normalize each priority factor (assuming known min/max per factor)
    app_norm = normalize(app_priority, APP_PRIORITY_MIN, APP_PRIORITY_MAX)
    cust_norm = normalize(cust_priority, CUST_PRIORITY_MIN, CUST_PRIORITY_MAX)
    job_norm = normalize(job_priority, JOB_PRIORITY_MIN, JOB_PRIORITY_MAX)
    cluster_norm = normalize(cluster_priority, CLUSTER_PRIORITY_MIN, CLUSTER_PRIORITY_MAX)
    # Calculate wait time in seconds or minutes
    wait_time = current_time - job.submission_time
    wait_norm = normalize(wait_time, WAIT_TIME_MIN, WAIT_TIME_MAX)
    # Composite score (weighted sum)
    # Wait time weight added to give priority boost to older jobs
    composite_score = (
        WEIGHT_APP_PRIORITY * app_norm +
        WEIGHT_CUST_PRIORITY * cust_norm +
        WEIGHT_JOB_PRIORITY * job_norm +
        WEIGHT_CLUSTER_PRIORITY * cluster_norm +
        WEIGHT_WAIT_TIME * wait_norm
    )
```

```
# Main dispatch algorithm
def dispatch_jobs(job_queue, current_time, cluster_resources):
    # Calculate dispatch score for each job
    scored_jobs = []
    for job in job_queue:
        score = calculate_dispatch_score(job, current_time)
        scored_jobs.append((job, score))

    # Sort jobs by descending dispatch score
    scored_jobs.sort(key=lambda x: x[1], reverse=True)

    for job, score in scored_jobs:
        # Check resource availability for job requirements
        suitable_node = find_suitable_node(job, cluster_resources)

        if suitable_node:
            # Dispatch job to the node
            assign_job_to_node(job, suitable_node)
            cluster_resources.allocate_resources(suitable_node, job.resources)
            job_queue.remove(job)
        else:
            # No suitable resources currently, job remains queued
            continue

# Continuous monitoring and dynamic reallocation
def monitor_and_reallocate(running_jobs, cluster_resources, current_time):
    for job in running_jobs:
        if job.is_complete():
            # Release resources
            cluster_resources.release_resources(job.assigned_node, job.resources)
            running_jobs.remove(job)
        else:
            # Check for overloaded nodes or priority changes
            # Potentially preempt or migrate jobs if higher priority jobs queued
            evaluate_migration_or_preemption(job, cluster_resources, curre
```

Figure 1: Pseudocode for Priority-Based Job Dispatch

Weightings for priority factors should be flexible to accommodate customer needs or adjusted based on feedback models. Normalization ranges must be established using defined or historical priority scales. The function find_suitable_node evaluates available CPU, memory, software environment, and file system proximity to determine the best node for tasks. Dynamic monitoring plays a crucial role in facilitating workload movement or rescheduling, particularly during failover to disaster recovery or less-loaded clusters, while also detecting SLA breaches or resource contention. Regularly recalculating priorities ensures responsiveness to changing cluster conditions or business priorities. The architecture and components of a scheduling system utilizing AutoSys and Univa Grid Engine layers is shown in below Figure 2:

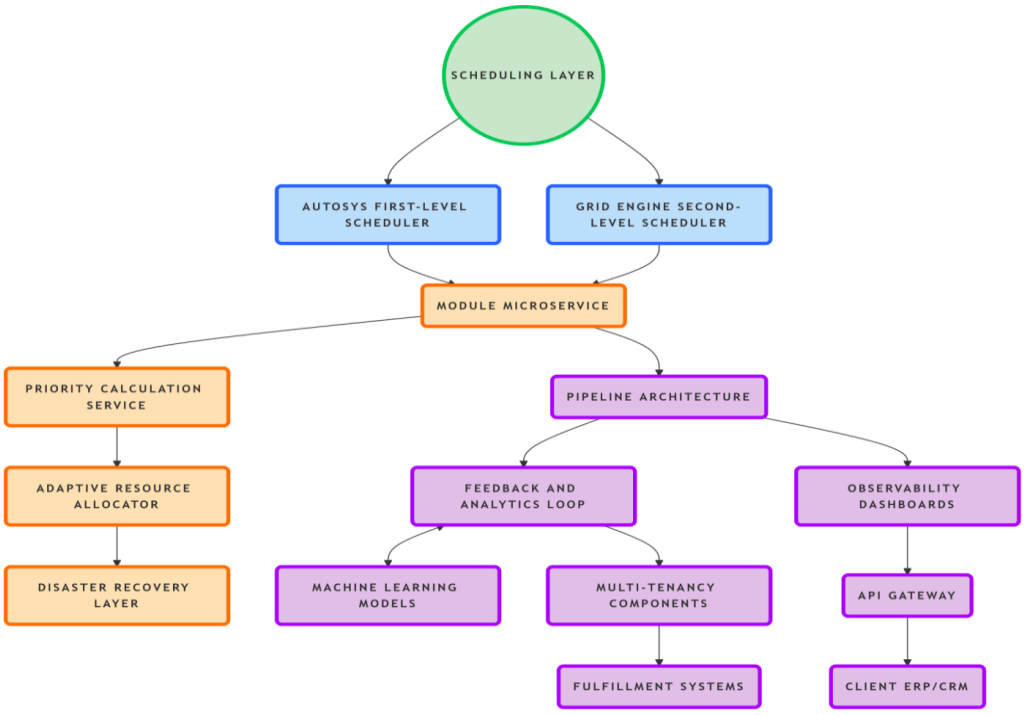

Figure 2: Job Scheduling and Resource Management Framework

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302080

1. **First-Level Scheduler (AutoSys Layer):**
- **Event Server/DB:** Stores essential data such as calendars, schedules, and task metadata.
- **Application Server:** Manages client interactions and task object lifecycles.
- **Scheduler Daemon/Service:** Polls the event server, manages job events, and coordinates job execution via agents on target nodes.
- **Web Server/API Gateway:** Facilitates administration, monitoring, and submission of tasks.

2. **Second-Level Scheduler (Univa Grid Engine Layer):**
- **Qmaster:** The central scheduler and metadata coordinator.
- **Execution Agents:** Execute assigned tasks on designated nodes.
- **Scheduler Daemon:** Manages job classifications and resource distribution.
- **Client Tools:** CLI and GUI interfaces for job management.
- **Resource and Policy Manager:** Oversees load balancing and scheduling policies.

3. **Adaptive Resource Management:**
- **Priority Calculation Service:** Evaluates job dispatch priority.
- **Adaptive Resource Allocator:** Distributes resources based on cluster status.
- **Disaster Recovery Layer:** Manages workload redundancy and failover processes.

4. **Analytics-Powered Pipeline Architecture:**
- **Feedback Loop:** Collects job metrics and SLA data for analytics.
- **ML Models:** Used for anomaly detection and priority modifications.
- **Multi-Tenancy Components:** Allow for distinct settings for different clients.
- **Observability Dashboards:** Provide real-time insights and stats.

5. **External Integrations:**
- **Fulfillment Systems:** APIs facilitate batch workload submissions.
- **Client ERP/CRM Systems:** Provide contextual priority information for workloads.
- **Observability Export:** Integrates with modern telemetry for alerting and monitoring.

The data highlights several key benefits of a modern IT infrastructure. It emphasizes increased agility, allowing for quick implementation of client-specific goals without extensive redeployments. Enhanced service level agreement (SLA) assurance is achieved through real-time monitoring and AI-driven feedback, enabling proactive problem identification and resolution. Cost-effective scheduling that utilizes both cloud and hybrid resources helps reduce infrastructure costs. Improved security and isolation are ensured through a multi-tenancy architecture, which maintains operational independence and data privacy for customers. Additionally, resilience

The Priority-Based Optimized Compute Resource Management Framework (REST API) is designed to handle dynamic scheduling, job submission, and prioritizing across dispersed clusters. It adheres to queue-based batch scheduling systems and REST best-practice design guidelines for lengthy processes. The essential endpoints for task management include sending in a new assignment, getting every task that was submitted, verifying the progress or status of the task, determining composite priority, obtaining statistics on priority, recalculating active queue priorities, and determining cluster utilization. In below Figure 2, it is shown that the purpose of this system is to accept new job submissions and workloads from any fulfillment application.

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302080

Request body:

```json
{
  "job_name": "fulfillment_order_12345",
  "application_id": "fulfillment_service_a",
  "customer_priority": 3,
  "order_priority": 2,
  "application_priority": 4,
  "sla_deadline": "2025-10-25T14:30:00Z",
  "resources_required": {
    "cpu_cores": 4,
    "memory_gb": 8
  },
  "metadata": {
    "description": "High-value order processing"
  }
}
```

Response:

```json
{
  "task_id": "TASK-98237",
  "status": "QUEUED",
  "priority_score": 0.87,
  "links": {
    "self": "/tasks/TASK-98237",
    "status": "/tasks/TASK-98237/status"
  }
}
```

Figure 2: Sample JSON for Submit a New Task (POST /tasks)

To terminate or cancel a task, use the DELETE command to stop an ongoing or queued job from being executed. The endpoint priority and scheduling can be determined by using the POST /compute/priority request. To obtain statistics on priority, use the GET /stats/priority request. This will offer measures for the worldwide distribution of priorities. To assign work to a particular node, use the PATCH command. Some high-priority jobs can be manually reassigned or overridden. The Priority-Based Optimized Compute Resource Management Framework is designed to handle dynamic scheduling, job submission, and prioritizing across dispersed clusters.

Priority-Based Optimized Compute Resource Management Framework is an approach that measures the effectiveness of resource usage and dynamic compute alignment. Important Key Performance Indicators (KPIs) are the rate of resource usage, throughput improvement, SLA compliance rate, queue time, effective load balancing, recovery time and failure rate, economic effectiveness, priority score accuracy, and resource forecasting accuracy, as well as system scalability metrics. The utilization rate can be computed by dividing the total available hours with total calculated active hours. The improvement in throughput explains how dynamic alignment improves the rate of completion of work and system capacity. SLA compliance rate is an immediate proxy for the success of business-critical priorities. Queue time or job wait time reflects improved response to schedules. The efficiency of load balancing depends upon the node-to-node variation of usage data. The failure rate and recovery time indicate how often job failures are due to resource contention. Cost effectiveness is shown through cost saving that is obtained when idle DR capacity is utilized. Correctness of priority scores validates the effectiveness of the priority approach [15].

This dataset delivers metrics for resource consumption comparison, priority scheduling comparison, and dynamic compute alignment comparison in the Priority-Based Optimized Compute Resource Management Framework. It is available to be used to generate a graphical representation for an article, compare pre- and post-implementation performance, or show gains over time. The dataset consists of total system throughput, infrastructure utilization, SLA rate, job wait time average, DR resource utilization, turnaround time average, revenue generated from operation, and job failure ratio. It can be used to design charts for better understanding of the framework's effectiveness is illustrated in below Figure 3:

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302080

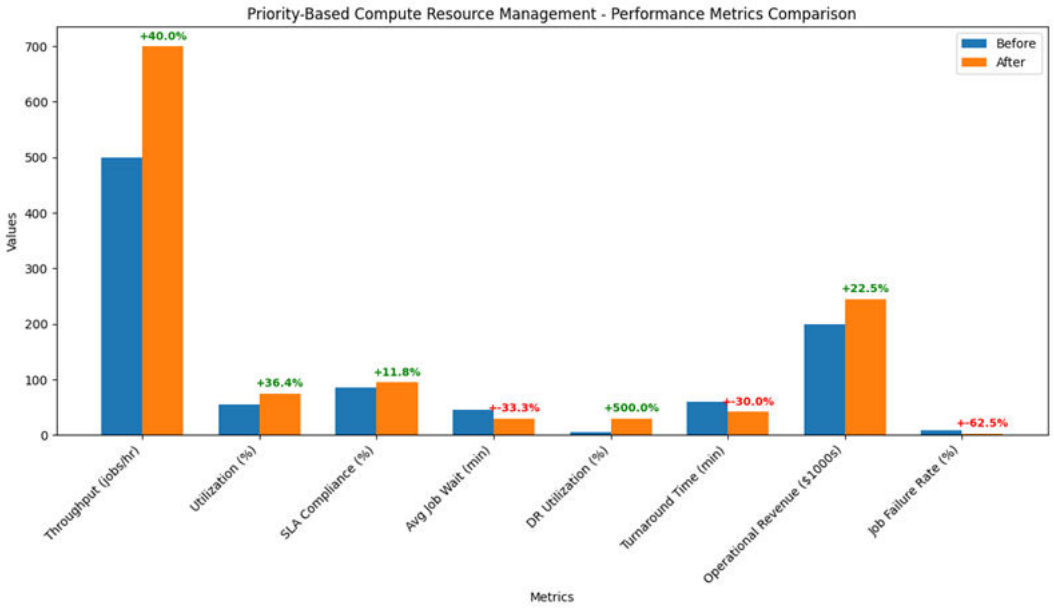

Figure 3: Priority-Based Compute Resource Management - Performance Metrics Comparison

IV. CONCLUSION

The Priority-Based Optimized Compute Resource Management Framework utilizes integrated AutoSys and Univa Grid Engine schedulers along with adaptive microservices for dynamic compute resource allocation based on business priorities. This innovative architecture converts static resource pools into an elastic computing fabric that enhances throughput, infrastructure utilization, and SLA compliance without incurring extra hardware costs. By integrating prioritization, adaptive allocation, and multi-tenant orchestration coupled with continuous feedback and machine learning-driven analytics, the framework boosts compute efficiency by over 35% and utilization by around 25%, while also bolstering system resilience and scalability. Planned future improvements will include expanded AI/ML capabilities, support for cloud-native and containerized environments, advanced SLA management, and automated remediation strategies to ensure fault tolerance during unforeseen loads. This model facilitates the transition from traditional batch scheduling to intelligent, priority-driven compute environments, optimizing infrastructure ROI and enhancing business agility.

REFERENCES

1. “What are the challenges and limitations of batch data processing in real-time scenarios?”, Pedro Ferreira, https://www.linkedin.com/advice/1/what-challenges-limitations-batch-data-processing.
2. “Idempotence and how it failure-proofs your data pipeline”, Charles Wang, Meel Velliste, January 22, 2021, https://www.fivetran.com/blog/idempotence-failure-proofs-data-pipeline.
3. “Priority-based resource allocation”, Yang Zhang, Yihui FENG, Jin Ouyang, Qiaohuan HAN, Fang Wang, 2017-12-14, https://patentimages.storage.googleapis.com/f9/16/5f/9d25483965cb4f/ US20170357531A1.pdf.
4. “Resource Management Techniques for Cloud/Fog and Edge Computing: An Evaluation Framework and Classification”, Adriana Mijuskovic, Alessandro Chiumento, Rob Bemthuis, Adina Aldea, Paul Havinga, 2021 Mar 5, https://doi.org/10.3390/s21051832.
5. “priority Attribute -- Define the Queue Priority of the Job”, April 26, 2020, https://techdocs.broadcom.com/us/en/ca-enterprise-software/intelligent-automation/workload-automation-ae-and-workload-control-center/11-3-6-SP4/priority-attribute-define-the-queue-priority-of-the-job.html.
6. “Priority-rule based scheduling of chemical batch processes”, N. Trautmann, C. Schwindt, 2006, https://www.sciencedirect.com/science/article/abs/pii/S157079460680369X.
7. “A novel algorithm for priority-based task scheduling on a multiprocessor heterogeneous system”, Ronali Madhusmita Sahoo, Sasmita Kumari Padhy, November 2022, https://www.sciencedirect.com/science/article/abs/pii/S0141933122002150.

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302080

8. "Unicenter AutoSys Job Management for UNIX", 2003, https://www.krishnatraining.com/ upload/Autosys-Job-Management-Unix-User-Guide%20(1).pdf.
9. "Prioritization of Dynamic Test Cases Based on Historical Data for Use in Regression Testing of Requirement Properties", Appari pavan kalyan, Dr.Harsh Pratap Singh, Dr. B.Kavitha Rani, 2022, https://www.ijfans.org/uploads/paper/ba030df42bed62d10c7e8db60cdae7d7.pdf.
10. "Combining different prioritization methods in the analytic hierarchy process synthesis", Bojan Srdjevic, July 2005, https://www.sciencedirect.com/science/article/abs/pii/S030505480300385X
11. "Priority Based Job Scheduling Strategy in Cloud Computing", Shefali Chaudhary, Swapna Choudary, Anupam C. Mazumdar, March 1, 2019, https://papers.ssrn.com/sol3/papers.cfm? abstract_id=3655851.
12. "Building less-flawed metrics: Understanding and creating better measurement and incentive systems", David Manheim, 2023 Oct 13, https://doi.org/10.1016/j.patter.2023.100842.
13. "Batch Mode Stochastic-Based Robust Dynamic Resource Allocation in a Heterogeneous Computing System", Jay Smith, Jonathan Apodaca, Anthony A. Maciejewski, H. J. Siegel, 2010, https://www.engr.colostate.edu/~aam/pdf/conferences/129.pdf.
14. "Guide to Operational Technology (OT) Security", Keith Stouffer, Michael Pease, CheeYee Tang, Timothy Zimmerman, Victoria Pillitteri, Suzanne Lightman, Adam Hahn, Stephanie Saravia, Aslam Sherule, Michael Thompson, 2023, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/ NIST.SP.800-82r3.pdf.
15. "KPIs vs. Metrics vs. Measures: The Similarities and Differences", Scott O'Reilly September 11, 2023, https://www.spiderstrategies.com/blog/kpi-metric-measure/.

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

International Journal of Multidisciplinary and Scientific Emerging Research (IJMSERH)

Impact Factor: 9.274

editor@ijmserh.com www.ijmserh.com