

Received 31 May 2026, accepted 14 June 2026, date of publication 17 June 2026, date of current version 24 June 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3704628

RESEARCH ARTICLE

Transport-Aware Performance Modeling for Big Data Workloads in Network-Constrained Hybrid Cloud Systems

ATTILA KANTO^{1,2}, AND ATTILA CSABA MAROSI^{3,4}

¹Doctoral School of Applied Informatics and Applied Mathematics, Óbuda University, 1034 Budapest, Hungary

²Cloudera, 1051 Budapest, Hungary

³HUN-REN Institute for Computer Science and Control (HUN-REN SZTAKI), Hungarian Research Network, 1111 Budapest, Hungary

⁴John von Neumann Faculty of Informatics, Óbuda University, 1034 Budapest, Hungary

Corresponding author: Attila Kanto (akanto@stud.uni-obuda.hu)

This work was supported in part by the Hungarian project no. TKP2021-NVA-01, implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021-NVA funding scheme.

ABSTRACT Accurate execution-time prediction for big data workloads in hybrid cloud environments is essential for offloading decisions at submission time. However, prediction remains challenging due to the nonlinear effects of network transport between on-premises clusters and public cloud resources, particularly bandwidth constraints and interconnect propagation latency. Existing approaches are often not suitable for submission-time offloading decisions: they either assume fixed infrastructure conditions without explicitly modeling transport effects, or require target-side pilot executions before making predictions. To address this problem, we present a transport-aware performance modeling framework for execution-time prediction in network-constrained hybrid clouds. By decoupling network transport effects from application logic and leveraging historical on-premises execution profiles, the framework enables execution-time estimation without per-query pilot runs on the target environment. The framework combines a stage-level analytical model whose components are derived from network transport principles (bandwidth-limited throughput, RTT-bound storage latency, and TCP slow-start dynamics) with a dependency-aware critical-path aggregation to derive end-to-end job runtime predictions. Its three structural coefficients are calibrated using Non-Negative Least Squares (NNLS) to preserve physical interpretability. The framework is evaluated using the TPC-DS benchmark on Apache Spark across eleven emulated hybrid network configurations, covering 2–25 Gb/s bandwidth and 0.3–25 ms round-trip time. Across these heterogeneous network conditions, the model achieves consistently high explanatory power, with $R^2 > 0.95$ at the stage level and $R^2 > 0.98$ at the job level. The gains are most pronounced under jointly constrained bandwidth and latency conditions. In the most severe configuration evaluated (2 Gb/s, 25.46 ms), the proposed framework reduces the Mean Absolute Error (MAE) of execution-time predictions by 76.8% (from 28.20 s to 6.54 s), while increasing job-level R^2 from 0.6625 to 0.9858 compared to a transport-agnostic baseline.

INDEX TERMS Big data, cloud bursting, hybrid cloud, network-constrained systems, performance modeling, system identification, transport-aware modeling.

I. INTRODUCTION

Cloud computing adoption has increased in recent years, offering elastic on-demand resources without upfront hardware investment. However, many organizations still

The associate editor coordinating the review of this manuscript and approving it for publication was Hosam El-Ocla¹.

operate large on-premises data platforms, and migrating all of their big data workloads to the public cloud is often uneconomical [1]. The risk of relying on infrastructure beyond organizational control remains a significant deterrent, reinforced by recent large-scale public cloud outages that caused global service disruptions [2], [3]. Existing

investments in private infrastructure, combined with data residency requirements, compliance, and integration of legacy systems, keep a large portion of data processing still on-premises [4], [5]. While these on-premises environments efficiently handle steady workloads, they often struggle with fluctuating resource demands, such as seasonal variations or periodic processing peaks. For example, in large-scale enterprise data platforms, workload arrival rates can be highly bursty, with peak submission rates reaching up to two orders of magnitude above the median load. Prior empirical studies of production MapReduce clusters report peak-to-median ratios as high as 260:1 [6]. Consequently, organizations face a trade-off between over-provisioning resources for rarely reached peaks and adopting flexible hybrid architectures. This strategy, known as cloud bursting or task offloading, allows enterprises to partially shift big data workloads, typically handled by distributed processing engines [7], [8], [9], [10] to the public cloud, while the offloaded workloads access data that remains on-premises. Although this approach provides substantial flexibility, it introduces a critical performance bottleneck: limited network bandwidth and increased latency between private and public environments [11].

Performance modeling and prediction for distributed data processing engines have been active areas of research for decades. Existing studies have focused on modeling factors such as CPU cores, data size, and system configuration [12], [13], [14], [15], [16], [17], [18]. Unlike compute or memory resources, which can be elastically scaled in the cloud, the interconnect network remains comparatively inflexible and is frequently the primary source of constraints [11]. A recent production-scale study further shows that, even under optimized hybrid placement strategies, remote data access during query execution remains substantial in practice [5]. Although that study focuses primarily on cost-aware placement, it also highlights that network transport-related effects remain practically relevant in large-scale hybrid analytics. As a result, accurately predicting the network's impact on job offloading is critical to hybrid cloud performance modeling.

Current performance prediction models face a fundamental trade-off that makes them unsuitable for cloud bursting decisions at submission time. In this context, submission-time offloading refers to making job placement decisions at submission without requiring per-query pilot runs, remote resource provisioning, or target-side performance measurements. Machine learning (ML) prediction models achieve high accuracy within the training distribution, but struggle to extrapolate to unseen operational scales or hardware configurations [16]. Conversely, high-fidelity analytical models can accurately estimate job execution time, but require per-query pilot runs on the target infrastructure [13], [15]. Moreover, existing models generally either ignore the network or treat network constraints as static parameters, representing network conditions as fixed inputs rather than modeling the

dynamic transport-layer behavior that governs data transfer performance.

To address these limitations, this paper investigates the feasibility of accurate, submission-time execution prediction for big data workloads in network-constrained hybrid cloud systems. We propose a transport-aware performance modeling framework that enables job execution-time prediction at submission time for hybrid cloud bursting. Our approach decouples network transport dynamics from application logic by leveraging historical on-premises execution profiles, without requiring per-query pilot runs on the target environment. In this paper, we make the following contributions:

- **Execution signature profiling methodology:** We introduce a technique to construct execution profiles from historical on-premises logs. These signatures serve as a robust baseline for projecting performance onto unseen hybrid infrastructures.
- **Transport-aware stage modeling:** We propose an analytical model operating at the granularity of execution stages, which are distinct sets of homogeneous parallel tasks separated by shuffle boundaries. By explicitly incorporating transport characteristics (latency, slow-start, congestion window saturation, and protocol overhead) into these boundaries, we isolate and quantify the impact of network constraints on data-intensive phases.
- **Physically interpretable system identification:** We calibrate the three structural coefficients of the analytical transport model using Non-Negative Least Squares (NNLS), a form of constrained linear regression applied to historical execution data. The non-negativity constraint enforces physical consistency and ensures that the calibrated model remains interpretable.
- **Resource-aware critical path aggregation:** We develop a closed-form aggregation model that bridges stage-level performance estimates to end-to-end job runtime. Our method accounts for parallel execution branches and resource contention, without the computational overhead of discrete-event simulations.

The framework takes historical on-premises stage execution profiles and hybrid interconnect parameters (available bandwidth and round-trip time) as inputs, and produces predicted per-stage and job-level execution times for the target hybrid cloud environment.

The framework is primarily targeted at recurring, read-intensive workloads with stable execution patterns, evaluated on Apache Spark [7] with HDFS-based [19] remote storage over emulated hybrid interconnects. Generalization to storage backends with distinct request and metadata models, such as cloud-native object stores [20], [21], or to highly dynamic, streaming, and ad hoc workloads is outside the current scope. Unlike workload forecasting, which predicts job arrival patterns, our framework estimates execution time for a specific submission from historical on-premises profiles and measured network parameters. Historical executions of recurring templates serve as independent observations in a regression setting.

To support reproducibility and further research, we intend to publicly release the benchmark dataset and the reference implementation of the proposed transport-aware framework [22].

The remainder of this paper is organized as follows. Section II reviews related work in hybrid cloud architecture and performance modeling. Section III introduces our transport-aware model and the underlying equations that characterize interconnect bottlenecks. Section IV describes the modeling framework, detailing the decoupled profiling architecture and the system identification methodology required for stage-level prediction. Section V describes our experimental setup that emulates realistic network constraints and a data-collection pipeline to obtain on-premises profiles. Section VI quantifies the empirical impact of remote data access on workload performance across diverse network regimes. Section VII rigorously evaluates the model's predictive accuracy and out-of-distribution generalization. Section VIII discusses broader implications and practical considerations. Finally, Section IX summarizes our findings and outlines future directions.

II. RELATED WORK

In this section, we examine the literature on hybrid cloud architectures, job offloading strategies, and performance modeling frameworks.

A. HYBRID CLOUD ARCHITECTURES AND CLOUD BURSTING

The existing literature can be systematically classified according to the degree of coupling between on-premises and cloud resources, distinguishing between architectures that form a unified logical cluster and those that maintain separate logical clusters. In the unified logical cluster architecture [23], [24], [25], [26], the components of the cluster span multiple sites over the Wide Area Network (WAN), forcing intermediate data (e.g., shuffle traffic) to traverse the interconnect. In the separate logical cluster approach [5], [27], workloads are routed or placed such that execution remains local to a specific site, with the interconnect primarily used for input/output data transfer.

In the first model, the hybrid cloud is treated as a unified pool of resources, where private- and public-cloud nodes form a monolithic cluster. While this simplifies cluster management, it exposes the framework's internal communication mechanisms to the WAN links. Empirical evaluations consistently highlight inter-site communication as the primary performance barrier in this architecture. This performance pattern was already observed by Roman et al. [28], who showed that, unlike in single-site deployments, where CPU is often the dominant constraint, network effects become the main limitation in hybrid Spark deployments, particularly during shuffle. Similar conclusions were reported by Ullah et al. [25] in hybrid OpenStack [29] and Azure [30] environments. Their results confirmed that execution time increases proportionally with the number of nodes borrowed

from the public cloud. They attributed this behavior to the substantial disparity between bandwidth available within the cloud environment and bandwidth available across the inter-cloud link. Rajput et al. [26] adopt the unified logical cluster model and address interconnect limitations using heuristic strategies that treat inter-cloud bandwidth as a scarce resource, thereby optimizing task placement to minimize the volume of intermediate data traversing the WAN. However, these studies do not explicitly model the impact of network constraints on execution time.

Verma et al. [31] study performance variability caused by hardware heterogeneity within single-cluster Hadoop deployments. They propose an adaptive node-oriented data placement strategy that mitigates straggler effects by improving data locality using historical execution logs. However, this approach assumes a co-located execution environment, and it does not consider hybrid cloud or cloud bursting scenarios, where data and compute are decoupled and execution time is dominated by network transport.

In contrast to monolithic cluster approaches, recent studies focus on separate logical clusters and job-placement-centric designs that treat on-premises and cloud sites as distinct execution environments. Moirai [5], based on Uber's multi-petabyte Presto and Spark deployments, represents the current state of the art in hybrid-cloud job and data placement. Moirai addresses key challenges in data egress, replication costs, and limited cross-site bandwidth using a Mixed-Integer Programming (MIP) optimization technique. By jointly optimizing job placement and data replication at both query and table granularity, it decides how to partition data, which tables to replicate, and where to route jobs to minimize overall costs. Moirai reduced the costs by an order of magnitude compared to prior methods such as Alibaba's Yugong [27]. However, Moirai's evaluation highlights the persistent challenges of hybrid analytics: even after optimization, their production-scale workloads still generate 421-751 TB of cross-site traffic per week. More than 95% of this traffic originates from remote data access during query execution, and the remaining 5% is attributed to data replication. Mendes and Almeida [32] propose a cost-driven entity migration recommender for hybrid data lakes that jointly optimizes storage, compute, and egress costs to determine the most cost-effective placement for data entities. While this work highlights the financial importance of data localization decisions in hybrid environments, it does not address submission-time execution-time prediction under explicit network transport constraints.

These findings show that while cost-centric frameworks significantly reduce data movement, cross-site processing remains unavoidable in real-world deployments. Remote data access remains an inherent characteristic of large-scale hybrid analytics. This highlights the critical need for our research, which complements Moirai's and Yugong's cost-optimization efforts by focusing on the performance dimension of remote data access. Specifically, we focus on the modeling and prediction of execution-time degradation

attributable to the network interconnect in scenarios where offloaded workloads perform remote data access over bandwidth- or latency-constrained hybrid communication links.

B. TRANSPORT-AGNOSTIC PREDICTION MODELS

A substantial body of prior research has investigated the prediction of execution time for Apache Spark and other big data workloads. Existing approaches can be broadly classified into two main categories: analytical models and machine learning-based models.

Analytical models formalize system behavior through mathematical expressions, enabling efficient evaluation using numerical methods [13]. PERIDOT [15] derives performance predictions by inferring job-specific model parameters from two reference executions of differing scales and subsequently constructing an application execution model based on these parameters. This framework relies on the assumption that the application's Directed Acyclic Graph (DAG) dependency structure remains consistent regardless of input data size or core count changes. More recent analytical work proposes a scalable Stochastic Activity Network (SAN) formulation that models interference across multiple concurrent YARN queues [34]. Tariq and Das extend deterministic modeling by flattening Spark's multi-job DAG into a monolithic graph and applying a graph-traversal execution algorithm [17]. These studies show that analytical approaches can provide accurate and interpretable predictions, but they typically rely on reference executions or micro benchmarks and do not explicitly address hybrid transport effects.

In machine learning-based models, researchers employ regression algorithms or neural networks to predict application performance using historical or experimental data. These approaches are commonly categorized as closed-box or gray-box models. The distinction between these models lies in the degree to which internal system knowledge or application details are incorporated into the model's features. Closed-box models rely on features available a priori (before execution starts), such as input data size and the number of executor cores. Gray-box models incorporate a richer set of features available a posteriori, capturing details of application execution, system internals, or resource allocation strategies. Ernest [12] defines a parametric structure based on known distributed computing behavior and relies on data-driven techniques to populate it, using NNLS regression to learn the coefficients of predefined features. Ahmed et al. [16] compare Kernel Ridge Regression and Gradient Boosting with analytical models and find that ML models achieve higher predictive accuracy within the training range, while analytical models perform substantially better for extrapolation beyond the training data. Gray-box approaches demonstrate improved prediction accuracy by incorporating Spark's stage-level metrics and resource allocation [14], [37]. Cheng et al. [33] address the challenge of Spark configuration optimization by proposing a performance prediction

framework based on AdaBoost. Their approach constructs stage-level performance models to estimate application runtime under different Spark configurations. Shen et al. [35] propose a deep learning-based framework for Spark performance prediction that combines attention and convolutional components to capture dependencies among configuration parameters, input data characteristics, and workload features. A recent study introduces a prediction framework that evaluates seven regression models to estimate job execution time [18]. Although Gradient Boosting achieves the highest accuracy in their evaluation setup, the framework remains fundamentally compute-centric and limited to a single-cluster environment, without accounting for network-link characteristics.

Garraalda-Barrio et al. [36] propose an adaptive incremental transfer learning framework for predicting workload execution times. Their approach integrates supervised and unsupervised learning to incrementally adapt performance models to new workloads and configuration settings. This approach primarily focuses on performance tuning for previously unseen workloads using a reference workload descriptor derived from an initial pilot execution. Their framework remains inherently tied to the training environment's infrastructure and lacks the transport-awareness required for generalization across hybrid cloud networks.

Across these works, two fundamental limitations remain. First, most existing prediction techniques require on-cluster execution or profiling to extract execution statistics or to run micro-benchmarks. Second, while prior models may account for bandwidth as a configuration parameter, they generally abstract the network as a static resource and do not explicitly model dynamic transport-layer effects. Consequently, their ability to extrapolate execution time across heterogeneous network environments, such as hybrid clouds, remains limited. This gap motivates the present work. We propose a transport-aware performance framework that generalizes execution-time degradation based on physical network parameters, eliminates the need for per-query remote execution, and leverages execution profiles collected from on-premises environments.

C. THEORY-GUIDED MODELS

The integration of domain knowledge into data-driven models has been formalized as Physics-Informed Neural Networks (PINNs), a term popularized by Raissi et al. [38] to embed physical laws into neural networks. This theory-guided approach has since expanded beyond neural networks and is referred to as Physics-Informed Machine Learning (PIML). In a comprehensive survey of the field, Karniadakis et al. [39] categorize these strategies into observational, learning, and inductive biases. The observational bias involves augmenting training data with synthetic samples derived from physical simulations. Learning bias modifies loss functions to penalize physically inconsistent predictions. Finally, inductive bias constrains the model architecture or feature space to

TABLE 1. Chronological summary and comparative analysis of performance prediction literature.

Paper (Year)	Methodology	Generalization Scope	Training Needs / Input Source	Benchmark	Transport Awareness
Venkataraman et al. (2016) [12]	Closed-box ML (NNLS)	Different dataset and cluster sizes	Small data samples using optimal experiment design	MLlib algorithms, GenBase and Speech pipeline	Mentioned, not modeled
Mustafa et al. (2018) [14]	Gray-box ML (LSBoost, NNLS, and MARS)	Different dataset, and cluster sizes	Offline training on feature vectors extracted from query stages	TPC-H, HIGGS dataset (MLlib)	No
Shah et al. (2019) [15]	Analytical	Different dataset, and cluster sizes	Two small reference executions per query	TPC-DS, text analytics, and ML workloads	No
Cheng et al. (2021) [33]	Gray-box ML (AdaBoost)	Configurations (up to 180+ parameters) and different input data sizes	Collected samples via projective sampling to reduce overhead	HiBench (six Spark benchmarks)	Mentioned, not modeled
Karimian-Aliabadi et al. (2023) [34]	Analytical (SAN + Fixed-Point Iteration)	Multi-queue / multi-user realistic scheduling, partitioned cluster capacity	Reference executions on a small cluster	TPC-DS	No
Shen et al. (2023) [35]	Deep Learning (MHAC - CNN + Attention)	Data scale, system parameters, and multi-task learning across applications	Historical logs	HiBench	No
Tariq and Das (2023) [17]	Analytical, Graph-based Model	Arbitrary input data sizes for fixed executor counts	Two small reference executions per query	TPC-DS (Query-64)	No
Farooq et al. (2025) [18]	ML (LR, DT, RF, XGB, SVR, MLP, and KNN)	Different configuration within the same cluster	Offline training on feature vectors extracted from job configuration and data size	WordCount and Average Word Length	No
Garralda-Barrio et al. (2025) [36]	Transfer Learning (NNLS)	Unseen workloads and evolving cluster configurations	Incremental, lazy learning from historical executions	HiBench	No
This work	Mechanistic System Identification (Analytical + NNLS)	Dynamic cross-cluster network environments	Transport parameters and historical logs	TPC-DS	Yes

Abbreviations: CNN = Convolutional Neural Network; DT = Decision Tree; GBRT = Gradient Boosted Regression Trees; KNN = k-Nearest Neighbors; LR = Linear Regression; LSBoost = Least-Squares Boosting; MARS = Multivariate Adaptive Regression Splines; MHAC = Multi-Head Attention Convolutional Network; MLP = Multilayer Perceptron; NNLS = Non-Negative Least Squares; RF = Random Forest; SAN = Stochastic Activity Network; SVR = Support Vector Regression; TPC = Transaction Processing Performance Council; XGB = Extreme Gradient Boosting.

strictly adhere to known physical laws. A recent review by Luo et al. [40] broadens this definition, highlighting that inductive biases can be effectively implemented through analytical feature augmentation.

While learning bias remains prevalent in continuous physical domains such as fluid dynamics, PINNs typically use automatic differentiation to guide training, ensuring that the learned model adheres to physical laws [41]. In this approach, physical consistency is a soft constraint on the learner rather than being resolved analytically. However, for

discrete computer systems, we argue that PIML is particularly valuable for system identification, not just for direct prediction. Consequently, our work follows a strict inductive bias strategy. The framework applies this strategy in two ways. First, the feature space is constructed from analytically derived, physically grounded transport terms that approximate key network-side mechanisms. Second, the structural coefficients are estimated using NNLS. This restricts the hypothesis space to physically valid, non-negative regions, ensuring that the model remains interpretable. In this way,

the framework combines theory-guided feature construction with constrained system identification.

The following section translates these theoretical principles into the analytical approximations that form the core of our performance modeling framework.

III. PROPOSED PERFORMANCE MODEL

This section details our transport-aware performance model. We begin by defining the operational boundaries and fundamental assumptions required to isolate the network interconnect as the primary independent variable influencing workload performance. Following this, we introduce the motivation for the stage-level abstraction and the task-wave concept, which serve as the structural foundations for our analytical components. We then derive mechanistic formulations for three additive components of stage execution time: bandwidth-limited data transfer, storage access latency, and dynamic TCP ramp-up effects. Finally, we describe the system identification methodology used to calibrate the model's coefficients, as well as the resource-aware aggregation model that integrates stage-level estimates into end-to-end job execution time prediction. The three scalar coefficients of these analytically derived components are calibrated via NNLS regression, as described in Section IV-C. The model structure is grounded in transport theory, and data is used solely to calibrate these three scalar coefficients. For clarity and ease of reference, Table 2 summarizes the principal symbols used throughout this section.

A. ASSUMPTIONS AND SCOPE

The primary objective of our transport-aware model is to enable fast, submission-time decision-making regarding job offloading. Existing performance models, as summarized in Table 1, are not designed to support this process because they either assume a homogeneous network environment or require target-side profiling. Such requirements make them unsuitable for scenarios in which the feasibility of cloud bursting must be determined a priori, before remote resources are provisioned or expenses incurred. Our model overcomes these limitations by eliminating the need for per-query target-side execution. Instead, it leverages existing local execution profiles and explicitly models the application's network interactions to predict cross-environment performance degradation.

Targeting the 56–68% of production workloads reported as repetitive [5], [42], [43], our model is designed to support offloading decisions for these stable, periodic pipelines, whereas ad hoc queries or new jobs without prior telemetry are explicitly out of scope. Prior work indicates that query planners and cost-based optimizers in big data processing engines are typically unaware of network characteristics [44], [45]. Consistent with this observation, we find that network parameters are not explicitly considered during logical and physical plan generation in our evaluation. Accordingly, the overall logical structure of execution plans remains largely preserved across on-premises and hybrid deployments. This

consistency enables the reuse of historical on-premises execution signatures as baseline inputs to model the performance impact of offloading workloads to the public cloud. Historical executions of a recurring template are independent observations, with network parameters as measured inputs rather than forecasted quantities.

Adaptive Query Execution (AQE) was enabled in all experiments, reflecting modern production deployments of Spark-based engines [46]. AQE optimizations (e.g., join re-selection and skew handling) are primarily driven by data-local runtime statistics. In our evaluation, we observed occasional plan variations across network regimes (e.g., minor differences in stage decomposition for some queries). However, these variations were limited in scope and did not materially affect prediction accuracy, and the proposed transport-aware model maintained high accuracy across all evaluated bandwidth-latency regimes.

We focus on read-intensive workloads in which data flows from on-premises storage to cloud-based compute, primarily incurring ingress rather than egress traffic. Expensive cloud egress fees can exceed compute costs for data-intensive workloads [5], [47], making ingress-dominant offloading more economically viable than workflows with substantial return traffic.

Throughout this work, HDFS [19] serves as the remote storage backend. Cloud-native object stores such as Amazon S3 [20] and Azure Data Lake Storage [21] differ from HDFS in their request and metadata access patterns: each partition access involves independent HTTPS requests, and clients typically employ persistent connection pools rather than per-task dedicated streams [48]. These characteristics affect both the storage-access latency model and the TCP connection mapping assumption, and extending the framework to object store backends remains outside the current scope. We further assume a one-to-one correspondence between active Spark tasks and TCP connections, where each task establishes a dedicated flow to the remote storage endpoint (Section III-D). This approximation holds for the HDFS single-stream block reads evaluated in this work and may weaken in configurations that employ connection pooling or storage-client connection reuse, in which case the effective number of independent TCP flows falls below the active task count.

To isolate the network bottleneck, we treat compute resources as a control variable, provisioning public cloud infrastructure that matches on-premises hardware specifications for CPU architecture, memory capacity, and total executor core count. Furthermore, we assume that the workload configuration remains unchanged and that concurrency, determined by the number of allocated executors and the number of cores specified at job submission, is constant. This approximates the steady-state behavior of a well-tuned dynamic application while eliminating the initialization latency of container acquisition. By keeping these compute and configuration parameters constant, we isolate the hybrid

TABLE 2. Summary of symbols used in the transport-aware performance model.

Symbol	Description	Unit
$\hat{T}_{\text{hybrid}}(S_i)$	Predicted execution time of stage S_i in a hybrid environment	s
$T_{\text{base}}(S_i)$	Observed on-premises baseline execution time of stage S_i	s
$T_{\text{bw}}(S_i)$	Predicted duration component limited by steady-state bandwidth	s
$T_{\text{storage}}(S_i)$	Predicted overhead component due to remote storage access latency	s
$T_{\text{dynamic}}(S_i)$	Predicted overhead component due to TCP slow-start dynamics	s
$\hat{T}_{\text{group}}(G_k)$	Predicted execution time of execution group G_k	s
$\hat{T}_{\text{job}}(\mathcal{J})$	Predicted total execution time of job $\mathcal{J}$	s
$N_{\text{tasks}}(S_i)$	Number of tasks in stage S_i	—
$N_{\text{executors}}$	Number of executors available to the application	—
N_{cores}	Number of cores per executor	—
$N_{\text{waves}}(S_i)$	Number of sequential task execution waves in stage S_i	—
$N_{\text{active}}(S_i)$	Active parallelism (concurrent tasks / TCP flows)	—
$N_{\text{partitions}}(S_i)$	Number of input partitions processed by stage S_i	—
$\rho(S_i)$	Partition density (average input splits processed per task)	—
B_{link}	Physical bandwidth capacity of the hybrid interconnect	bytes/s
B_{flow}	Steady-state throughput upper bound for a single TCP flow	bytes/s
$B_{\text{eff}}(S_i)$	Effective aggregate bandwidth available to stage S_i	bytes/s
$b_{\text{base}}(S_i)$	Baseline average data demand rate of stage S_i	bytes/s
$R_{\text{target}}(S_i)$	Target per-flow throughput required to saturate available bandwidth	bytes/s
W_{max}	Maximum receiver-advertised or congestion window size	bytes
W_{init}	Initial TCP congestion window size	bytes
$W_{\text{target}}(S_i)$	Congestion window size required to sustain R_{target}	bytes
RTT	Round-trip time of the hybrid interconnect	s
$\nu(S_i)$	TCP slow-start ramp-up time (analytical lower bound)	s
β_{bw}	Learned scaling coefficient capturing protocol and system-level inefficiencies	—
β_{storage}	Learned coefficient capturing structural and metadata access latency	—
β_{dynamic}	Learned coefficient capturing transient transport and congestion overheads	—
$\mathcal{J}$	Job dependency graph	—
S_i	i -th stage in job $\mathcal{J}$	—
G_k	Execution group of dependency-equivalent stages	—

network interconnect parameters as the primary independent variables influencing performance in our experimental setup.

B. WORKLOAD MODELING ABSTRACTION

Modern data-parallel processing engines, including Apache Spark and Hive on Tez, rely on a common Directed Acyclic Graph (DAG) abstraction to represent execution dependencies. As demonstrated by Karimian-Aliabadi et al. [13], these systems share equivalent structural semantics and resource-filling scheduling behaviors, suggesting that performance models derived from one framework are broadly applicable to the class of DAG-based engines. We select Apache Spark as the reference platform for validating our model because of its widespread industrial adoption and fine-grained execution metrics at the job, stage, and task levels, which enable rigorous empirical analysis.

Apache Spark [7] operates as a distributed, in-memory computing framework. Execution is coordinated by a Driver program, which interfaces with a resource manager to launch persistent Java Virtual Machine (JVM) processes called executors. Executors are deployed on worker nodes. The fundamental unit of data abstraction is the Resilient

Distributed Dataset (RDD), an immutable collection of objects partitioned across the cluster. The Driver compiles the execution logic into a DAG of stages. These stages are delineated by shuffle dependencies, which require data redistribution before subsequent dependent stages can consume the data.

At the granular level, each stage is decomposed into a set of independent tasks, each executing the same operation on different data partitions. Task scheduling follows a resource-filling paradigm constrained by the number of available executor cores. As the number of tasks in a stage typically exceeds the cluster's concurrency limit, the runtime processes tasks in sequential batches, a pattern formally defined as task waves by Shah et al. [15].

For our analytical model, we abstract this behavior by discretizing stage execution into a sequence of waves. The total number of waves, $N_{\text{waves}}(S_i)$, for stage S_i is determined by the ratio of tasks to the total available concurrency slots, formally:

$$N_{\text{waves}}(S_i) = \left\lceil \frac{N_{\text{tasks}}(S_i)}{N_{\text{executors}} \cdot N_{\text{cores}}} \right\rceil \quad (1)$$

By applying the ceiling function, (1) provides a unified formulation for both regimes: it correctly captures saturated scenarios of multi-wave execution, and non-saturated scenarios where the task count is less than the available maximum capacity ($N_{\text{tasks}}(S_i) < N_{\text{executors}} \cdot N_{\text{cores}}$).

Modeling performance at this granularity provides several advantages. It aligns naturally with the physical execution model, since each stage corresponds to a well-defined unit in the data-flow graph. This abstraction also isolates the network-dependent portion of execution: shuffle exchanges occur within the same network domain, while data reads from storage cross the inter-site interconnect between environments. From a practical standpoint, stage-level modeling yields a compact and interpretable dataset. Prior studies have shown that stage-level abstractions achieve high predictive accuracy and robustness across workloads [14], [15], [17], supporting our choice of stage-level granularity for performance modeling in distributed systems.

C. STAGE PREDICTION OVERVIEW

Our framework operates at the granularity of execution stages, which are distinct sets of homogeneous parallel tasks separated by shuffle boundaries. This abstraction defines a clear boundary: certain stages require transferring data remotely, whereas others read data exclusively within the local cluster. At the core of our stage-level predictor is a transport-aware analytical component informed by established network transport principles. Rather than modeling the full protocol stack explicitly, the model captures key transport effects, including bandwidth saturation, latency, and TCP slow-start behavior, through simplified analytical approximations. This abstraction provides a principled foundation for extrapolating performance across previously unseen network environments:

$$\hat{T}_{\text{hybrid}}(S_i) = T_{\text{bw}}(S_i) + T_{\text{storage}}(S_i) + T_{\text{dynamic}}(S_i) \quad (2)$$

$\hat{T}_{\text{hybrid}}(S_i)$ denotes the predicted execution time of the i -th execution stage, S_i , within a hybrid cloud environment. The total duration is decomposed into three additive components that capture distinct sources of execution cost. $T_{\text{bw}}(S_i)$ represents the bandwidth-limited data transfer time, derived by scaling the baseline on-premises execution profile according to the effective throughput of the hybrid interconnect. $T_{\text{storage}}(S_i)$ captures the storage access overheads incurred during remote I/O, including request latency, connection establishment and authentication overheads, and protocol-level processing costs that are paid independently of the transferred data volume. The $T_{\text{dynamic}}(S_i)$ models the transient transport overhead introduced by TCP slow-start behavior. In the following subsections, we derive the formulations for each component, beginning with the steady-state bandwidth constraints.

D. BANDWIDTH-LIMITED EXECUTION MODEL

In the absence of packet loss, TCP flow throughput B_{flow} is governed by the congestion window size, which grows until

it reaches the receiver's advertised window or the sender's congestion window limit W_{max} . Under these steady-state conditions, the theoretical upper bound on flow throughput is given by [49]:

$$B_{\text{flow}} = \frac{W_{\text{max}}}{RTT} \quad (3)$$

where RTT denotes the round-trip time, dominated in this ideal state by the path's fixed propagation delay.

When a Spark stage executes in the cloud and reads data from on-premises sources, the aggregated throughput is constrained by two physical limits: the total capacity of the interconnect (B_{link}) and the sum of concurrent TCP flows' throughput. We assume a one-to-one mapping between Spark tasks and TCP connections, where each active task establishes a distinct TCP flow to the remote data store. Therefore, the number of active flows (N_{active}) corresponds to the degree of active parallelism, bounded by either the task count or the maximum capacity:

$$N_{\text{active}}(S_i) = \min(N_{\text{tasks}}(S_i), N_{\text{executors}} \cdot N_{\text{cores}}) \quad (4)$$

Consequently, the effective available bandwidth (B_{eff}) is defined as the minimum of the physical link capacity and the aggregate window-limited throughput:

$$B_{\text{eff}}(S_i) = \min(B_{\text{link}}, N_{\text{active}}(S_i) \cdot B_{\text{flow}}) \quad (5)$$

The usable bandwidth is governed by whichever constraint is reached first: the physical capacity of the hybrid link, or the aggregate window-limited capacity that the concurrent TCP flows can collectively sustain. For example, with $W_{\text{max}} = 4$ MB and $RTT = 25$ ms, each flow sustains at most $B_{\text{flow}} = 160$ MB/s (1.28 Gb/s). Four concurrent flows yield an aggregate window-limited capacity of 5.12 Gb/s. On a 2 Gb/s hybrid link, the link saturates first and $B_{\text{eff}} = 2$ Gb/s; on a 25 Gb/s link, the same four flows cannot fill the pipe and $B_{\text{eff}} = 5.12$ Gb/s.

To quantify the performance impact without relying on CPU profiling, we adopt a bandwidth-dominant approximation. We treat the stage baseline average data demand rate $b_{\text{base}}(S_i)$, measured as the total volume processed divided by the baseline duration of the same stage $T_{\text{base}}(S_i)$. If the effective hybrid bandwidth (B_{eff}) falls below the baseline bandwidth (b_{base}), the stage is treated as strictly I/O-bound. In this regime, the bandwidth-limited execution time (T_{bw}) is modeled by scaling the baseline execution time by the corresponding bandwidth bottleneck ratio:

$$T_{\text{bw}}(S_i) = \beta_{\text{bw}} \cdot T_{\text{base}}(S_i) \cdot \max\left(1, \frac{b_{\text{base}}(S_i)}{B_{\text{eff}}(S_i)}\right) \quad (6)$$

This component enables the framework to adjust the on-premises baseline for the physical throughput limits of the hybrid link. With this, the T_{bw} captures the intuitive inverse relationship between effective throughput and execution time: as the link capacity decreases, data transfer time increases proportionally to the ratio $b_{\text{base}}/B_{\text{eff}}$. We also introduce the coefficient β_{bw} to account for

protocol- and system-level inefficiencies and implementation overheads, which we will determine using system identification explained in Section IV-C. The $\max(1, \dots)$ operator enforces a physical lower bound on execution time. It ensures that the predicted duration does not fall below the empirically observed baseline if the effective hybrid interconnect bandwidth B_{eff} exceeds the throughput required by the stage.

E. STORAGE ACCESS LATENCY

In distributed file systems such as HDFS, Amazon S3, and Azure Data Lake Storage, reading a dataset requires distinct Remote Procedure Calls (RPCs) or HTTPS requests to resolve file paths, retrieve block locations, and open input streams. Unlike data transfer, connection establishment and metadata operations are latency-bound rather than bandwidth-bound. In an architecture with disaggregated storage and compute, every file or block access incurs multiple round-trip penalties. The number of penalties depends on multiple factors, such as underlying storage implementation and security configuration.

We model this overhead as a function of the partition density (ρ), defined as the average number of input splits processed per task:

$$\rho(S_i) = \frac{N_{\text{partitions}}(S_i)}{N_{\text{tasks}}(S_i)} \quad (7)$$

Assuming the task processes its assigned partitions sequentially, each partition access incurs a latency penalty of several RTTs for connection establishment, authentication, and application-level handshakes. Within a single wave, the critical-path cost equals ρ sequential access penalties; across N_{waves} batches, the total storage-access overhead (T_{storage}) is:

$$T_{\text{storage}}(S_i) = \beta_{\text{storage}} \cdot N_{\text{waves}}(S_i) \cdot \rho(S_i) \cdot RTT \quad (8)$$

The coefficient β_{storage} quantifies the structural latency incurred per logical partition access. The coefficient represents the aggregate latency introduced across the communication stack. This encompasses the baseline overhead of the standard TCP three-way handshake, as well as the additional round-trip times required to establish authenticated sessions. The latter depends on the specific protocol version, and can be approximated as ≈ 2 RTTs for HDFS SASL/Kerberos, 2 RTTs for TLS 1.2, and 1 RTT for TLS 1.3. It also encapsulates application-layer access patterns in which the initialization sequence is often executed multiple times per partition. In columnar formats, the file footer serves as the mandatory entry point, containing zone maps with min-max statistics; the engine must perform initial round-trips to retrieve this metadata for selection pruning [50]. If the query predicate is satisfied, then the subsequent requests are sent to fetch the actual data body. This requires several discrete round-trips to storage through the high-latency interconnect. We acknowledge that β_{storage} represents a statistical expectation rather than a deterministic constant

for every individual data access. While reading the footer is mandatory, reading the data body is conditional.

F. DYNAMIC RAMP-UP EFFECTS

The component described in the previous section accounts for the storage interaction costs, and the steady-state bandwidth model introduced in Section III-D captures transfer time at the link's sustainable rate. They do not account for the transient ramp-up cost required to reach that rate. As the round-trip time increases, the amount of in-flight data required to fully utilize the link grows linearly, and is governed by the bandwidth-delay product. In practice, TCP employs a slow-start mechanism to probe available network capacity. It initializes with a conservative congestion window (W_{init}) and increases it based on congestion control algorithms as acknowledgments are received [51].

We model this transient behavior as an additive ramp-up overhead incurred for each storage access. First, we determine the target congestion window size (W_{target}) required to sustain the effective per-flow throughput. The target flow rate (R_{target}) is given by:

$$R_{\text{target}}(S_i) = \min\left(\frac{b_{\text{base}}(S_i)}{N_{\text{active}}(S_i)}, \frac{B_{\text{link}}}{N_{\text{active}}(S_i)}, B_{\text{flow}}\right) \quad (9)$$

and the corresponding target window is:

$$W_{\text{target}}(S_i) = R_{\text{target}}(S_i) \cdot RTT \quad (10)$$

Assuming ideal exponential growth of the congestion window, the slow-start ramp-up time required to reach the target window W_{target} is:

$$\nu(S_i) = \max\left(0, RTT \cdot \log_2\left(\frac{W_{\text{target}}(S_i)}{W_{\text{init}}}\right)\right) \quad (11)$$

The $\max(0, \cdot)$ operator ensures that the ramp-up penalty remains non-negative. When $W_{\text{target}} \leq W_{\text{init}}$, the initial window already meets the bandwidth-delay product requirement, and no additional slow-start ramp-up time is incurred. To estimate the total slow-start overhead, we scale the theoretical ramp-up time by the execution topology and the learnable β_{dynamic} coefficient:

$$T_{\text{dynamic}}(S_i) = \beta_{\text{dynamic}} \cdot N_{\text{waves}}(S_i) \cdot \rho(S_i) \cdot \nu(S_i) \quad (12)$$

This term captures the transient cost of reaching target throughput on each new connection. The ramp-up time ν is incurred per partition access per wave, so stages with frequent connection establishment over high-RTT paths accumulate the largest penalty. While the ideal exponential growth assumption establishes a theoretical lower bound, real-world latency is dominated by protocol safeguards. Modern implementations, such as TCP CUBIC, utilize Hybrid Slow-Start (HyStart) to prevent buffer overflows, often terminating the ramp-up phase prematurely [52]. Although analytical models can describe these dynamics explicitly [49], [53], [54], they rely on network-layer metrics such as instantaneous packet loss probability and retransmission timeouts. These low-level parameters and metrics are opaque to the big data

application-level logs. Consequently, rather than simulating these unobservable packet-level events, we encapsulate the aggregate transport inefficiencies, including premature slow-start exit, congestion avoidance, and retransmission overhead, into the learnable coefficient β_{dynamic} .

G. JOB-LEVEL AGGREGATION MODEL

To predict the total job execution time $\hat{T}_{\text{job}}$, we aggregate the individual stage predictions $\hat{T}_{\text{hybrid}}(S_i)$ by modeling the job's dependency graph. While earlier approaches approximated job duration as the cumulative sum of all stages, Mustafa et al. [14] correctly note that this ignores stage parallelism, yielding a loose upper bound rather than a realistic runtime estimate. Although detailed execution modeling can be achieved through discrete-event simulation [13], our objective is instead to reduce the systematic bias introduced by cumulative summation while preserving the low computational overhead required for submission-time prediction.

Big data processing engines execute stages concurrently when data dependencies allow and sufficient compute resources are available [7], [9]. To capture this behavior, we decompose the job graph $\mathcal{J}$ into a sequence of execution groups $\{G_1, G_2, \dots, G_m\}$, where each group represents a set of stages that can be scheduled concurrently. Formally, each group G_k is a subset of stages that share the same dependency level in the directed acyclic graph and therefore have no mutual dependencies.

The predicted duration of each group is determined to distinguish between three execution scenarios. First, for a group $G_k = \{S_i\}$ that contains a single stage, the duration of the group is simply the predicted duration of the stage:

$$\hat{T}_{\text{group}}(G_k) = \hat{T}_{\text{hybrid}}(S_i) \quad (13)$$

Second, when a group G_k contains multiple independent stages, and the cluster has sufficient executor cores to schedule them simultaneously, the group execution time is determined by the critical path, which is the longest-running stage in the set:

$$\hat{T}_{\text{group}}(G_k) = \max_{S_i \in G_k} \{\hat{T}_{\text{hybrid}}(S_i)\} \quad (14)$$

Third, when the aggregate core requirement of the stages in G_k exceeds the total available executor slots, resource contention forces the scheduler to queue stages. In this saturation regime, we approximate the group duration as the sum of its constituent stages:

$$\hat{T}_{\text{group}}(G_k) = \sum_{S_i \in G_k} \hat{T}_{\text{hybrid}}(S_i) \quad (15)$$

We acknowledge that this discrete grouping is a conservative approximation of the scheduler's behavior. While stage dependencies impose strict ordering constraints across groups, the execution engine does not enforce an additional artificial barrier beyond those dependencies. In practice, tasks from downstream stages may begin execution as soon as

their input partitions become available and resources are freed, resulting in limited inter-stage overlap. In the network-constrained hybrid configurations evaluated in this work, data access and transfer latencies constitute a substantial portion of total execution time, which reduces the impact of such fine-grained overlap on overall runtime prediction. Finally, the total job time $\hat{T}_{\text{job}}(\mathcal{J})$ is the cumulative sum of the predicted group durations, $\hat{T}_{\text{group}}(G_k)$:

$$\hat{T}_{\text{job}}(\mathcal{J}) = \sum_{k=1}^m \hat{T}_{\text{group}}(G_k) \quad (16)$$

Figure 1 illustrates the dependency-aware grouping on a simplified job graph. Nodes denote stages S_i , and dashed rectangles represent execution groups G_k , ordered according to the job's topological structure. Stages S_1, S_2 , and S_3 have no mutual shuffle dependencies and form G_1 , allowing potential parallel execution subject to resource availability. Stage S_4 , which depends on S_1 and S_2 , forms G_2 . Stage S_5 , dependent on S_3 and S_4 , defines G_3 , and finally S_6 , dependent on S_5 , forms G_4 .

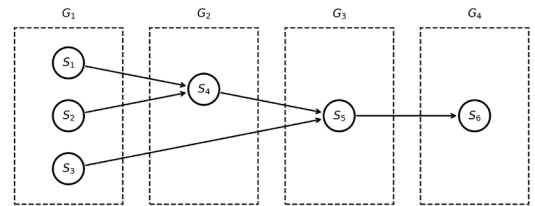


FIGURE 1. Example of dependency-aware stage grouping.

In summary, this section defined a stage-level model representing bandwidth, storage latency, and TCP dynamics. Combined with a dependency-aware aggregation strategy, this model captures the non-linear impact of network constraints on total job execution time. We now describe the system architecture that operationalizes these components.

IV. TRANSPORT-AWARE MODELING FRAMEWORK

The primary objective of our framework is to enable accurate, submission-time performance prediction for hybrid cloud bursting without requiring target-side profiling for every new workload. Our approach relies on the relative stability of logical query plans across environments, enabling on-premises telemetry to serve as a high-fidelity baseline for cloud execution. Our framework follows a decoupled, two-phase architecture.

A. PHASE 1: PROACTIVE ON-PREMISES PROFILING

The first phase involves collecting lightweight performance metrics during on-premises job execution to establish a performance baseline. The primary data sources consist of standard Spark event logs and granular stage-level metrics retrieved from the Spark History Server REST API. We developed a lightweight metrics interceptor that hooks

into the on-premises execution engine to extract and serialize performance counters into a structured JSON format.

A central requirement for our transport-aware prediction is the association of these metrics with specific job templates. In large-scale production environments, a job template is typically defined as the canonical logical plan of a query after literals, predicates, and constants are stripped. A stable identifier is then derived from this canonical plan using hashing methods [55]. Structural elements, such as table references, are preserved in the template definition. Because table definitions resolve to specific storage locations and associated metadata (e.g., file format and compression), queries executed over different storage configurations may yield distinct physical execution templates and therefore require separate execution profiles.

While our framework is compatible with canonical logical-plan hashing mechanisms, for this study we use the intrinsic query template IDs provided by the TPC-DS benchmark suite, combined with the storage format and compression settings, to uniquely identify execution templates. By assigning telemetry to these templates, we create a historical knowledge base that enables our model to extrapolate performance changes when the same logical plan is submitted. The overall profiling workflow is illustrated in Figure 2.

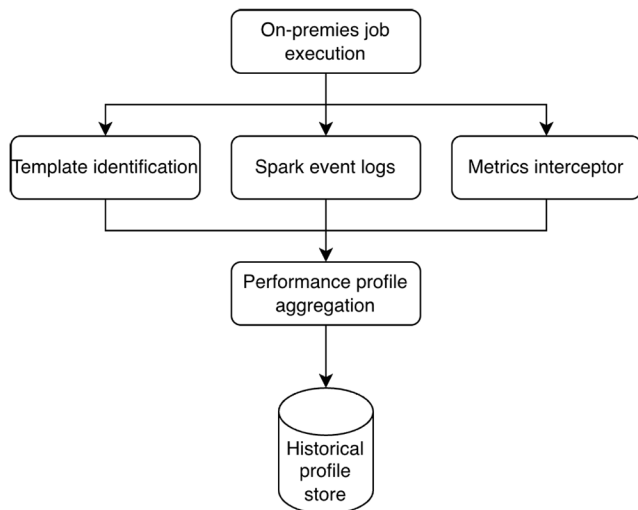


FIGURE 2. Proactive on-premises profiling workflow.

B. PHASE 2: REACTIVE SUBMISSION-TIME PREDICTION

The second phase occurs at the time of job submission and enables prediction-based offloading decisions. The complete workflow is illustrated in Figure 3. First, the framework identifies the incoming query's template to retrieve the corresponding historical on-premises profile and its stages. The estimation process follows a hierarchical logic, beginning with stage-level prediction described in Section III-C. Following this stage-level prediction, the framework employs the job-level aggregation model introduced in Section III-G to estimate the total job execution time. This phase accounts

for stage dependencies and potential overlap between stages to ensure the final estimate reflects the job's structural complexity. Finally, the system executes an automated job dispatch decision based on this aggregated prediction: if the estimated hybrid runtime meets the user-defined degradation threshold, the job is offloaded to the public cloud; otherwise, it is retained for local execution.

This workflow relies on the stability of historical profiles, which remain up to date as long as jobs execute on-premises. Consistent offloading, however, prevents new on-premises data from being collected, and the stored profile may grow stale. Profile age or a growing gap between predicted and observed execution times can indicate this condition, though addressing it would require either resuming on-premises execution or extending profile collection to cloud deployments. Integrating an online feedback loop for real-time model self-correction remains a subject for future investigation.

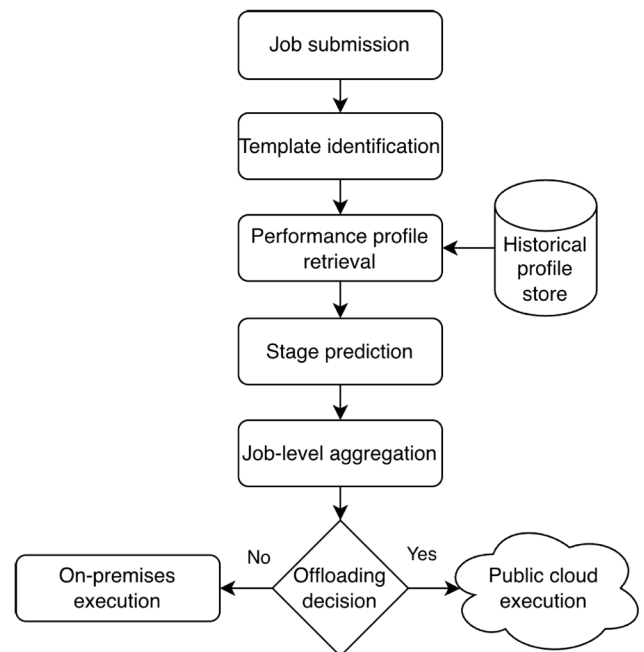


FIGURE 3. Submission-time prediction workflow.

C. SYSTEM IDENTIFICATION METHODOLOGY

Enabling submission-time prediction requires determining the β_{bw} , $\beta_{storage}$, and $\beta_{dynamic}$ coefficients without per-query pilot runs. We resolve this by decoupling infrastructure calibration from job-specific prediction. To identify these coefficients, we formulate parameter estimation as a Non-Negative Least Squares (NNLS) problem on a representative subset of workloads. This approach ensures that the estimated coefficients remain physically valid ($\beta \geq 0$), preventing the model from fitting negative transport-related overheads that would be physically impossible.

We determine the three structural coefficients in two steps. The first takes on-premises stage profiles from a representative calibration set and the corresponding network parameters

as input and substitutes them into the coefficient-free transport terms $T_{bw}^*(S_i)$, $T_{storage}^*(S_i)$, and $T_{dynamic}^*(S_i)$, derived from (6), (8), and (12), yielding the predictor matrix Φ as output. The second takes Φ and the observed hybrid execution times $\mathbf{y}$ of the stages in the calibration set as input and applies NNLS to yield exactly three scalar coefficients $\hat{\beta}$ as output. These coefficients are then applied at prediction time through (17) to any recurring query whose on-premises profile is available.

Based on the additive stage-level model in (2), we rewrite the prediction for stage S_i as a constrained linear combination of coefficient-free transport terms:

$$\hat{T}_{hybrid}(S_i) = \beta_{bw} \cdot T_{bw}^*(S_i) + \beta_{storage} \cdot T_{storage}^*(S_i) + \beta_{dynamic} \cdot T_{dynamic}^*(S_i) \quad (17)$$

Here, $T_{bw}^*(S_i)$, $T_{storage}^*(S_i)$, and $T_{dynamic}^*(S_i)$ denote the coefficient-free terms derived from the bandwidth-limited transfer model, the storage-access latency model, and the dynamic TCP ramp-up model, respectively. These terms are grounded in both the stage decomposition of distributed data processing engines and established network transport principles. The nonlinear effects are encoded analytically in the three model components; coefficient calibration is therefore formulated as a constrained linear least-squares problem.

Given a calibration set of m observed stages, we collect these terms into a predictor matrix Φ and the corresponding measured hybrid stage execution times into a target vector $\mathbf{y}$. The coefficient vector is defined as

$$\beta = \begin{bmatrix} \beta_{bw} \\ \beta_{storage} \\ \beta_{dynamic} \end{bmatrix} \quad (18)$$

The coefficient identification problem is formulated as the following NNLS optimization problem:

$$\hat{\beta} = \arg \min_{\beta} \|\Phi\beta - \mathbf{y}\|_2^2 \quad \text{subject to} \quad \beta \geq 0 \quad (19)$$

Here, $\Phi \in \mathbb{R}^{m \times 3}$ is the predictor matrix whose rows correspond to observed stage instances and whose columns correspond to the analytically derived transport terms, $\mathbf{y} \in \mathbb{R}^m$ is the vector of measured hybrid stage execution times, and $\beta \in \mathbb{R}^3$ is the vector of unknown transport coefficients. The objective minimizes the squared Euclidean norm of the residual vector, i.e., the sum of squared differences between predicted and measured stage execution times, under non-negativity constraints.

This formulation aligns with the mathematical philosophy of established distributed systems modeling frameworks, ranging from foundational scalability models such as Ernest [12] to recent adaptive transfer learning approaches for Spark task-level metrics [36]. While prior studies primarily use NNLS to characterize local resource scaling or task-level configurations, our framework adapts this interpretable approach to network transport in hybrid cloud environments. This system identification is performed offline

as an infrastructure-level calibration step that characterizes the persistent physical constraints of the link. Because these coefficients capture the system's physical and architectural properties rather than its query logic, they exhibit empirical stability across different query templates in our evaluation.

We acknowledge that these coefficients serve a dual role: they quantify the explicitly modeled mechanics while also acting as structural proxies for systemic variables not captured in our formal equations, such as hardware-specific metadata processing or security protocol overheads.

D. DEPLOYMENT CONSIDERATIONS

The framework requires no specialized infrastructure beyond a telemetry store for historical stage profiles and for measuring the current hybrid interconnect parameters. The profiling phase (Section IV-A) is non-intrusive, reads from the Spark History Server without modifying job execution, and can therefore be enabled in production without impacting running workloads. At submission time, the prediction reduces to a small number of arithmetic operations on pre-computed stage profiles, making its latency negligible within a submission-time decision window. The dispatch decision is a threshold comparison against a user-defined degradation tolerance, which is straightforward to configure and audit. This phase integrates naturally with workflow orchestrators such as Apache Airflow [56] as a pre-execution operator, enabling automated offloading decisions within existing pipeline definitions. Alternatively, in a broader deployment context, the predicted execution times could serve as runtime estimates for higher-level resource managers such as Starburst [57]. Such systems, which mediate between on-premises and dynamically provisioned cloud clusters, could incorporate these estimates to drive more informed placement decisions based on predicted performance overhead.

In summary, the framework follows a decoupled structure that separates proactive profiling and system identification from the prediction phase. Workload signatures and coefficients are determined beforehand. At the moment of job submission, the system only executes the prediction workflow to make informed offloading decisions. The following section details the experimental environment and the emulated hybrid interconnect used to evaluate the model's predictive accuracy across the TPC-DS workload under varying network constraints.

V. EXPERIMENTAL SETUP

To empirically validate the proposed model, we orchestrated a synthetic hybrid cloud topology comprising two logically distinct clusters. While both environments were physically deployed on Amazon Web Services (AWS) infrastructure to ensure hardware homogeneity and reproducibility, their logical configurations were strictly separated to emulate a real-world hybrid deployment.

A. CLUSTER TOPOLOGY AND CONFIGURATION

The first cluster represents the on-premises site and stores data in HDFS. This choice reflects the continued prevalence of HDFS-based data platforms in enterprise environments. Industry surveys report that on-premises deployments collectively manage tens of exabytes of data [58]. Given the operational and economic constraints associated with data gravity, a substantial portion of enterprise data remains resident in on-premises storage systems, making HDFS a representative baseline for evaluating hybrid cloud bursting scenarios. The second cluster serves as the public cloud execution engine, representing the ephemeral clusters handling offloaded jobs. This cluster accesses input data remotely from the HDFS storage of the first cluster via the emulated hybrid interconnect, while processing intermediate shuffle data using local ephemeral storage.

The overall experimental topology, including the Traffic Shaping Gateway (TSG) used to emulate the hybrid interconnect by enforcing bandwidth and latency constraints, is illustrated in Figure 4.

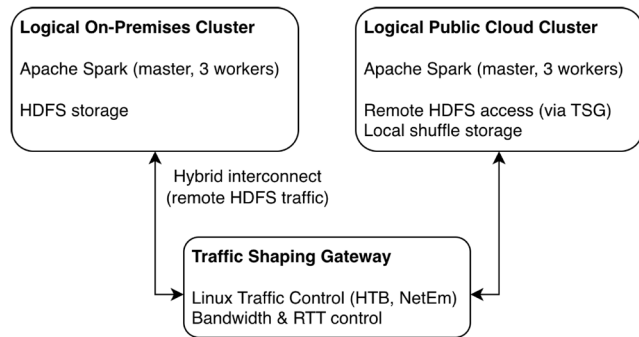


FIGURE 4. Experimental hybrid cloud topology with emulated interconnect.

The experimental environment was built on a production-grade Cloudera Data Platform (CDP) version 7.3.1.100 [59], integrating Apache Spark 3.4.1 and Hadoop 3.1.1. Both clusters were configured with an identical topology consisting of a single leader node dedicated to cluster management services and three worker nodes responsible for benchmark execution. The detailed hardware specifications are summarized in Table 3.

TABLE 3. Experimental cluster hardware configuration.

Specification	Leader Node	Worker Node (x3)
Instance Type	r5.4xlarge	m6i.4xlarge
vCPUs	16	16
Memory (GB)	128	64
Data Disk Type	-	GP3 EBS
Number of Disks	-	4
Disk Capacity (TB)	-	1 (per disk)
Disk Throughput (MB/s)	-	250 (per disk)
Baseline Network (Gb/s)	5.0	6.25
Burst Network (Gb/s)	10.0	12.5

B. NETWORK INTERCONNECT EMULATION

To isolate the impact of hybrid interconnect performance on query execution, we established a deterministic network

emulation environment. This setup enables precise control over network parameters. We implemented a dedicated TSG, logically positioned between the on-premises and public-cloud clusters. To force all inter-site traffic through this control point, we configured static policy-based routing on all cluster nodes.

The TSG implements traffic shaping using the token bucket algorithm, one of the most widely used rate limiting techniques [60]. Our specific implementation leverages the Hierarchical Token Bucket (HTB) [61] queuing discipline provided by the Linux kernel's traffic control (tc) subsystem [62]. While this mechanism is primarily used to govern inter-cluster traffic rates, we also applied rate limiting on every node to eliminate the burst network credit mechanism typical of AWS EC2 instances [63]. We thereby neutralize these transient increases in network capacity by imposing a per-node strict throughput ceiling equal to the baseline network value listed in Table 3.

To extend the TSG framework's capabilities established in our prior work [11], we integrated latency constraints using the Linux kernel's NetEm module [64]. This allows the TSG configuration to synthesize the artificial propagation delays characteristic of geographically distributed hybrid clouds. In our experiments, we inject three different round-trip propagation delays: 0 ms, 10 ms, and 25 ms. To ensure the accuracy of the emulated environment, we empirically verified these constraints through ICMP-based measurements, resulting in observed average RTTs of 0.301 ms, 10.492 ms, and 25.456 ms, respectively:

- **0 ms (observed 0.301 ms):** Serves as the baseline, representing a near-ideal local connection or a dedicated fiber link with minimal propagation delay.
- **10 ms (observed 10.492 ms):** Simulates the latency characteristic of a high-quality regional or metropolitan area connection.
- **25 ms (observed 25.456 ms):** Represents the latency typical of a hybrid cloud interconnect between regions.

The discrepancy between the configured delays and the measured RTT values is attributable to the baseline latency of the underlying infrastructure, including physical propagation delay, switch processing time, and virtualization overhead. These factors add to the delay introduced by NetEm.

The bandwidth constraints were aligned with industry-standard AWS Direct Connect [65] tiers. The TSG enforced four bandwidth caps (2, 5, 10, and 25 Gb/s) across three latency profiles, yielding twelve possible configurations. One combination (2 Gb/s at 10 ms) was not measured, resulting in eleven evaluated network environments. The 25 Gb/s configuration was intentionally selected to exceed the aggregate storage throughput capacity of the worker nodes. As the combined disk bandwidth is below this limit, the hybrid interconnect does not impose a throughput constraint under these conditions, allowing us to approximate a transport-unconstrained baseline.

C. BENCHMARK SELECTION

For evaluation, most existing studies use HiBench [66] or TPC-DS [67] benchmarks, as summarized in Table 1. The TPC-DS is particularly compelling for our research because its read-heavy nature aligns with the economic asymmetry of hybrid cloud networking. This benchmark models a retail business environment characterized by periodic surges in data volume and query load, such as seasonal sales or promotional campaigns. These scenarios reflect a common motivation for hybrid cloud bursting, where enterprises temporarily extend on-premises deployments to the public cloud to handle transient demand peaks.

Our experimental validation uses a fixed TPC-DS scale factor of 1,000, which generates approximately 1 TB of raw data. We systematically evaluate the impact of transport-layer constraints across diverse network configurations and workload characteristics. During the evaluation, we vary the storage format and compression strategy:

- **Format variability:** We consider two columnar storage formats: (i) ORC and (ii) Parquet, which represent the de facto standards for high-performance analytics [50].
- **Compression strategy:** We consider three configurations: (i) uncompressed, maximizing transferred data volume; (ii) Snappy, providing a balanced I/O-CPU tradeoff; and (iii) Gzip (Parquet only), shifting the workload toward a CPU-intensive regime by reducing transferred data volume while increasing computational overhead [11].

This experimental design results in 495 distinct execution templates (99 TPC-DS query templates evaluated across five storage/compression configurations), spanning diverse execution regimes. Within this set, query runtimes range from sub-second executions transferring only a few megabytes to complex analytical workloads accessing more than 150 GB of data across over 2,100 HDFS blocks. By systematically varying the compression-to-compute ratio, the model is evaluated under both I/O-bound and CPU-bound conditions.

D. EVALUATION METRICS

To assess the prediction accuracy of the transport-aware model, we use statistical metrics that quantify the deviation between predictions ($\hat{y}_i$) and ground truth values (y_i), where y_i is either the duration of an execution stage ($T_{\text{hybrid}}(S_i)$) or the end-to-end job execution time (T_{job}).

Mean Absolute Error (MAE) measures the average size of prediction errors, ignoring their direction. It applies equal weight to all errors, making it robust to outliers.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (20)$$

Root Mean Square Error (RMSE) is a quadratic scoring rule that measures the average magnitude of the error by squaring the differences before averaging. This approach

penalizes larger deviations more heavily.

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (21)$$

Mean Absolute Percentage Error (MAPE) is a standard measure of prediction accuracy expressed as a percentage. However, it has limitations for short-lived queries and execution stages, where negligible absolute errors can lead to disproportionately large percentage deviations due to inherent system noise.

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right| \cdot 100\% \quad (22)$$

To address MAPE's limitations, we use the Weighted Mean Absolute Percentage Error (WMAPE). By normalizing absolute error by the total magnitude of observed values, WMAPE gives more weight to long-duration executions, reducing distortion from short stages. This mirrors practical cloud cost priorities, where accurate predictions for long-running jobs matter more economically than similar relative errors in short tasks.

$$\text{WMAPE} = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{\sum_{i=1}^n |y_i|} \cdot 100\% \quad (23)$$

The Coefficient of Determination (R^2) quantifies the proportion of variance in the observed outcomes explained by the model predictions. Values closer to 1 indicate stronger explanatory power.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (24)$$

These metrics assess prediction quality and are reported across all evaluated network configurations in the model evaluation tables in Section VII. With the deterministic emulation environment and evaluation metrics established, we proceed to characterize the empirical impact of network constraints on the TPC-DS workload.

VI. IMPACT OF REMOTE DATA ACCESS

To quantify the impact of accessing data through the hybrid interconnect, we use the Execution Time Degradation (ET_r) metric established in our previous work [11]. This metric represents the relative increase in job execution time in the cloud compared to a local on-premises baseline:

$$ET_r = \frac{T_{\text{cloud}} - T_{\text{on-premises}}}{T_{\text{on-premises}}} \cdot 100\% \quad (25)$$

where T_{cloud} denotes the measured execution time of a job when executed on the cloud compute cluster with remote data

access over the hybrid interconnect, and $T_{\text{on-premises}}$ denotes the execution time of the same job executed locally on the on-premises cluster with data locality. Our experimental results confirm that degradation is primarily associated with two distinct network regimes: throughput saturation and latency-induced protocol throttling. These impacts are summarized across the evaluated network configurations in Table 4.

TABLE 4. Execution time degradation (ET_r) across varying hybrid interconnect conditions.

Bandwidth	Latency	Total Time	Median ET_r	Total ET_r
<i>On-Premises (baseline)</i>		5.07 h	0.0%	0.0%
25 Gb/s	0.30 ms	5.14 h	2.13%	1.41%
25 Gb/s	10.49 ms	5.94 h	28.22%	17.17%
25 Gb/s	25.46 ms	7.58 h	78.94%	49.41%
10 Gb/s	0.30 ms	5.19 h	2.07%	2.39%
10 Gb/s	10.49 ms	6.01 h	31.38%	18.59%
10 Gb/s	25.46 ms	7.68 h	81.16%	51.32%
5 Gb/s	0.30 ms	5.74 h	7.93%	13.10%
5 Gb/s	10.49 ms	6.33 h	38.56%	24.85%
5 Gb/s	25.46 ms	7.87 h	92.17%	55.10%
2 Gb/s	0.30 ms	7.93 h	40.70%	56.40%
2 Gb/s	25.46 ms	11.52 h	120.13%	127.30%

The aggregate results show a non-uniform relationship between network constraints and workload performance. In the 25 Gb/s and 0.30 ms regime, which approximates a transport-unconstrained baseline, a slight degradation relative to the on-premises configuration persists. This performance penalty results from reduced data locality. On-premises HDFS supports local short-circuit reads, whereas in the hybrid cloud setup, compute and storage are disaggregated, requiring all input data read from HDFS to traverse the network.

Analysis of the 25 Gb/s and 10 Gb/s regimes shows that the median ET_r generally exceeds the total ET_r , particularly under higher latency conditions. This suggests that at higher bandwidths, the aggregate degradation is primarily driven by fixed transport-layer and protocol overheads, which disproportionately affect shorter tasks. However, when bandwidth is restricted to 2 Gb/s, the total ET_r (127.30%) exceeds the median ET_r (120.13%). In this regime, the physical throughput limit becomes the dominant bottleneck for I/O-bound queries, which account for the majority of the cumulative execution time.

Propagation delay introduces an additional bottleneck even when bandwidth is abundant. Under the 25 Gb/s configuration, increasing the round-trip time to 25.46 ms results in a 49.41% increase in total benchmark execution time. This behavior reflects latency-induced transport inefficiencies, where extended RTT prolongs TCP congestion window growth and amplifies per-request protocol overheads.

Figure 5 illustrates the distribution of Execution Time Degradation (ET_r) for 495 TPC-DS query executions under a single hybrid network configuration. The diagram corresponds to the 2 Gb/s bandwidth limit with 25.46 ms round-trip time. The degradation distribution is highly skewed and forms a pronounced long tail, driven by highly network-sensitive

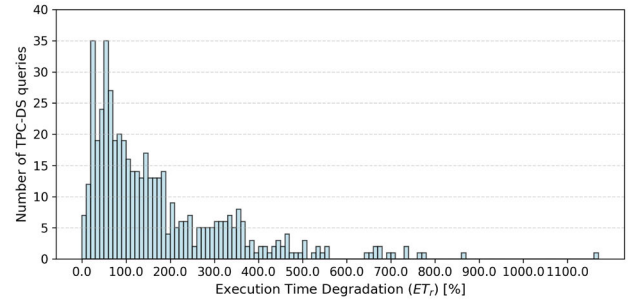


FIGURE 5. Distribution of Execution Time Degradation (ET_r) across 495 TPC-DS queries under a fixed network configuration (2 Gb/s, 25.46 ms RTT).

workloads such as Query28 [68] and Query85 [69], which are discussed later in this section. The median degradation under this configuration is 120.13%, indicating that more than half of the evaluated queries experience at least a doubling of execution time relative to the on-premises baseline.

To understand the mechanics of the evaluated network regimes, we analyze individual query behaviors. In throughput-limited scenarios, queries with high data-to-compute ratios become bandwidth-bound because a substantial fraction of their execution time is spent transferring input data rather than performing computation. For example, Query28 [68], which exhibits the highest data transfer volume in our evaluation of the TPC-DS suite (approximately 151.55 GB), experiences a degradation of 1142.11% in the 2 Gb/s and 0.30 ms configuration. With this configuration, CPU utilization drops from 80.08% to 5.32%, indicating that the majority of execution time is spent stalled on remote data transfer. Under high-bandwidth but high-latency conditions (25 Gb/s and 25.46 ms RTT), Query28 also exhibits substantial degradation. In this regime, the slowdown is primarily attributable to bandwidth-delay product effects: the increased RTT prolongs TCP congestion window growth and delays convergence to the steady-state window size required to saturate the link. Consequently, execution time increases from 52.12 s to 196.68 s, corresponding to an ET_r of 277.40%.

Certain workloads exhibit high sensitivity to latency regardless of data volume. Under the 25 Gb/s and 25.46 ms RTT configuration, Query85 [69] is heavily impacted by the cumulative effect of round-trip time. It processes 2,184 partitions of the `web_returns` table and 1,823 partitions of the `web_sales` table while moving 3.36 GB of data, which is approximately 2.22% of the HDFS input data read by Query28. As round-trip time increases, the cumulative wait time for these thousands of small HDFS accesses, including TCP and security handshakes, as well as footer metadata reads from the underlying columnar files, inflates the execution duration. Consequently, execution time increases from 6.40 s to 53.00 s, corresponding to a 728.13% increase.

A critical finding of our characterization is that aggregate network I/O volume alone is not a reliable indicator of execution-time degradation in hybrid environments. Instead,

degradation is primarily driven by the volume of input data accessed remotely across the hybrid interconnect, whereas internal shuffle traffic within the cloud cluster has limited impact. Consider Query67 [70], which processes nearly 40 GB of data. While 35 GB is moved during the shuffle phase, this traffic utilizes local ephemeral storage and does not traverse the constrained hybrid interconnect. Despite a long absolute execution time (450.47 s), Query67 exhibits negligible degradation (3.01%) under 2 Gb/s and 25.46 ms RTT constraints. Its bottleneck is low parallelism during query processing, not transport-layer limitations.

These empirical findings reinforce the need for a transport-aware approach that enables selective, submission-time offloading decisions, identifying workloads that are likely to benefit from cloud bursting while avoiding those for which remote execution would incur substantial degradation.

VII. MODEL EVALUATION

This section evaluates the proposed transport-aware framework through a multi-stage validation process. We begin by performing system identification to calibrate the structural coefficients (β). We then assess the accuracy of the stage-level predictor by comparing it with transport-agnostic baselines and quantifying the precision gains from the bandwidth- and latency-aware components of the model. Furthermore, we evaluate the stage predictor's robustness to variations in maximum transmission unit (MTU) size and assess its out-of-distribution generalization capability across unseen network environments. Finally, we evaluate the resource-aware aggregation model to assess the framework's effectiveness in predicting end-to-end job execution times. In all evaluation tables throughout this section, bandwidth and latency are the controlled independent variables.

A. TRANSPORT-AGNOSTIC BASELINES

As a baseline, we assess a representative transport-agnostic predictor. Representative models in this class, including Ernest [12], Mustafa et al. [14], Ahmed et al. [16], and Farooq et al. [18], are listed in Table 1. Although such models can achieve high accuracy within a specific deployment environment, they omit bandwidth and latency as explicit input features, and they are not designed to generalize execution time across heterogeneous network conditions in hybrid cloud settings.

To define a transport-agnostic reference predictor, we assume that hybrid execution time equals the on-premises runtime ($\hat{T}_{\text{hybrid}} = T_{\text{base}}$). This formulation reflects the infrastructure-invariance assumption that arises when network characteristics are not explicitly modeled: in the absence of transport-aware variables, hybrid execution is implicitly treated as equivalent to baseline behavior. Under favorable network conditions where bandwidth is abundant and latency is low (25 Gb/s, 0.30 ms), this reference predictor achieves high explanatory power $R^2 > 0.98$ at the job level, as shown in Table 14. In such regimes, network effects are secondary to local disk and CPU characteristics, and hybrid

performance closely matches on-premises behavior. Rather than picking one specific model from this class, we adopt $\hat{T}_{\text{hybrid}} = T_{\text{base}}$ as a representative baseline, which constitutes the best achievable predictor for any transport-agnostic model in a network-constrained hybrid setting. Without bandwidth and RTT as input features, no such model can distinguish hybrid execution from on-premises behavior.

1) STAGES WITHOUT REMOTE DATA TRANSFER

We first consider stages that do not perform remote data access across the hybrid interconnect. The prediction results of the transport-agnostic model are summarized in Table 5. Across all evaluated bandwidth-latency configurations, prediction accuracy remains consistently high ($R^2 > 0.99$, MAE < 0.30 s).

These high R^2 values reflect the near-equivalence between hybrid and on-premises execution in the absence of remote data transfer. In this regime, transport effects are effectively absent, and hybrid execution closely matches the baseline runtime. The prediction task therefore reduces to modeling minor runtime variability rather than transport-induced degradation. As such, this setting does not constitute a challenging test of transport sensitivity.

TABLE 5. Transport-agnostic baseline prediction accuracy on stages without remote data transfer.

Bandwidth	Latency	MAE	RMSE	WMAPE	R^2
25 Gb/s	0.30 ms	0.22	1.29	4.33%	0.9966
25 Gb/s	10.49 ms	0.24	1.18	4.65%	0.9972
25 Gb/s	25.46 ms	0.27	1.16	5.30%	0.9973
10 Gb/s	0.30 ms	0.21	1.05	4.10%	0.9977
10 Gb/s	10.49 ms	0.20	0.85	4.02%	0.9985
10 Gb/s	25.46 ms	0.27	1.22	5.26%	0.9970
5 Gb/s	0.30 ms	0.24	1.43	4.74%	0.9959
5 Gb/s	10.49 ms	0.20	0.80	3.98%	0.9986
5 Gb/s	25.46 ms	0.27	1.95	5.21%	0.9919

2) STAGES WITH REMOTE DATA TRANSFER

We evaluate the same transport-agnostic baseline on stages that transfer data across the hybrid interconnect. The results are summarized in Table 6.

Under favorable network conditions (e.g., 25 Gb/s, 0.30 ms), explanatory power remains high ($R^2 \approx 0.99$). In this regime, hybrid execution closely follows the on-premises baseline, as the network does not constitute the primary bottleneck. However, as bandwidth decreases and latency increases, prediction accuracy deteriorates substantially. Under constrained conditions (e.g., 5 Gb/s, 25.46 ms), R^2 drops to 0.5618 and error metrics increase significantly. In these network-bound regimes, the baseline systematically underestimates execution time because transport-layer effects are not modeled.

For clarity of incremental comparison, we restrict the subsequent analysis to bandwidth values between 5 and 25 Gb/s. Lower bandwidth constraints (2 Gb/s) are introduced in Section VII-C3 to evaluate out-of-distribution extrapolation performance.

TABLE 6. Transport-agnostic baseline prediction accuracy on stages with remote data transfer.

Bandwidth	Latency	MAE	RMSE	WMAPE	R ²
25 Gb/s	0.30 ms	0.20	0.77	7.56%	0.9893
25 Gb/s	10.49 ms	1.13	2.53	31.12%	0.9047
25 Gb/s	25.46 ms	3.33	7.43	56.43%	0.6132
10 Gb/s	0.30 ms	0.25	1.02	9.25%	0.9814
10 Gb/s	10.49 ms	1.20	2.73	31.92%	0.9016
10 Gb/s	25.46 ms	3.45	7.68	57.22%	0.6031
5 Gb/s	0.30 ms	0.80	3.34	23.98%	0.8625
5 Gb/s	10.49 ms	1.65	4.32	39.32%	0.8039
5 Gb/s	25.46 ms	3.73	8.50	59.22%	0.5618

Overall, these results demonstrate that transport-agnostic models perform well when network effects are negligible or secondary to local disk throughput or CPU constraints. Once hybrid interconnect becomes a limiting factor, the accuracy of the prediction deteriorates systematically.

B. COMPONENT-WISE EVALUATION OF THE TRANSPORT-AWARE STAGE-LEVEL PREDICTOR

To isolate the contribution of individual transport effects, we evaluate the stage-level predictor component-wise. Specifically, we first analyze the bandwidth-dependent transfer term and then incorporate the latency-driven storage and dynamic transport components. This decomposition allows us to quantify how distinct physical mechanisms influence predictive accuracy.

In this section, we restrict the analysis to stages that perform remote data transfers. As demonstrated in Section VII-A1, stages without remote I/O exhibit negligible sensitivity to network conditions and therefore do not require transport-aware adjustment.

1) BANDWIDTH-DEPENDENT COMPONENT

Table 7 reports the prediction accuracy of the bandwidth-dependent component of our transport-aware model when only the bandwidth-scaling formulation (6) is applied. In this configuration, the T_{storage} and T_{dynamic} terms are disabled, isolating the contribution of T_{bw} . Compared to the transport-agnostic baseline, this component yields substantial improvements in bandwidth-constrained regimes.

TABLE 7. Bandwidth-dependent component prediction accuracy on stages with remote data transfer.

Bandwidth	Latency	MAE	RMSE	WMAPE	R ²
25 Gb/s	0.30 ms	0.19	0.77	7.31%	0.9893
25 Gb/s	10.49 ms	1.08	2.44	29.63%	0.9117
25 Gb/s	25.46 ms	3.27	7.32	55.42%	0.6237
10 Gb/s	0.30 ms	0.18	0.62	6.83%	0.9932
10 Gb/s	10.49 ms	1.09	2.43	28.86%	0.9217
10 Gb/s	25.46 ms	3.33	7.40	55.23%	0.6312
5 Gb/s	0.30 ms	0.22	0.59	6.56%	0.9957
5 Gb/s	10.49 ms	1.02	2.25	24.32%	0.9466
5 Gb/s	25.46 ms	3.09	6.77	49.15%	0.7222

At 5 Gb/s and 0.30 ms RTT, R^2 increases from 0.8625 in the transport-agnostic setting to 0.9957, while MAE improves from 0.80 s to 0.22 s. Similar improvements are observed across other low-latency configurations. These results provide evidence that when RTT is low, throughput

limitation is the dominant transport effect, and the scaling relationship expressed by (6) captures the primary source of execution-time deviation.

However, as RTT increases, the predictive power of the bandwidth-only formulation diminishes. At 10.49 ms RTT, R^2 falls into the 0.9117–0.9466 range, and at 25.46 ms RTT, it further degrades to 0.6237–0.7222. In these high-latency settings, prediction accuracy remains comparable to the transport-agnostic baseline. This behavior is consistent with the model design: the bandwidth term captures throughput constraints, but it does not represent RTT-driven storage and congestion effects.

2) LATENCY-DRIVEN TRANSPORT COMPONENTS

Within our transport-aware model, we now evaluate the full stage-level formulation by incorporating the latency-driven components in addition to the bandwidth-dependent term. Specifically, the T_{storage} and T_{dynamic} terms are activated alongside T_{bw} , thereby forming the complete transport-aware stage-level predictor.

As shown in Table 8, the inclusion of these terms yields substantial improvements under higher-latency conditions. At 25.46 ms RTT, R^2 increases from approximately 0.6237–0.7222 in the bandwidth-only model to between 0.9618 and 0.9672. The observed improvement in evaluation metrics suggests that the RTT-driven mechanisms account for the majority of the variance unexplained by the bandwidth-only formulation. By explicitly modeling these latency-sensitive components, the transport-aware predictor achieves high accuracy across heterogeneous bandwidth-latency configurations.

TABLE 8. Latency-driven transport component prediction accuracy on stages with remote data transfer.

Bandwidth	Latency	MAE	RMSE	WMAPE	R ²
25 Gb/s	0.30 ms	0.20	0.78	7.62%	0.9889
25 Gb/s	10.49 ms	0.36	0.93	9.81%	0.9871
25 Gb/s	25.46 ms	0.84	2.22	14.17%	0.9655
10 Gb/s	0.30 ms	0.19	0.63	7.01%	0.9929
10 Gb/s	10.49 ms	0.37	0.93	9.75%	0.9886
10 Gb/s	25.46 ms	0.88	2.38	14.62%	0.9618
5 Gb/s	0.30 ms	0.21	0.58	6.44%	0.9958
5 Gb/s	10.49 ms	0.45	1.13	10.63%	0.9866
5 Gb/s	25.46 ms	0.94	2.33	14.87%	0.9672

The component-wise evaluation highlights the complementary roles of the modeled transport mechanisms. Bandwidth scaling corrects the dominant error in throughput-limited settings but leaves considerable error under higher RTT conditions. Incorporating the latency-driven storage and dynamic transport components reduces this remaining error and improves prediction accuracy across all evaluated network configurations.

C. SENSITIVITY AND STABILITY ANALYSIS OF STRUCTURAL COEFFICIENTS

In this subsection, we evaluate the structural reliability of the transport-aware model to ensure that the learned coefficients

represent physical system properties rather than dataset-specific artifacts. We first examine the statistical stability of the parameters across diverse workload samples using bootstrapping, followed by a workload-level cross-validation of coefficient stability across different selections of query templates. We then assess the model's ability to generalize to unseen network environments (out-of-distribution), evaluate the sensitivity of predictions to coefficient miscalibration, and examine its sensitivity to link-layer configurations, such as MTU variations.

1) STATISTICAL STABILITY AND PARAMETER ROBUSTNESS

The transport-aware modeling framework introduces a small set of structural coefficients that require empirical calibration, as described in Section IV-C. In this section, we evaluate the statistical stability and robustness of these parameters. Coefficient estimation is performed using NNLS regression to enforce non-negativity constraints. To assess robustness, we perform a stratified bootstrap procedure with 1,000 iterations. In each iteration, stages are sampled with replacement while preserving the distribution of baseline input data volume. Stages are partitioned into five quantile bins based on `input_read_gb_baseline`, representing the volume of HDFS input data read during baseline execution as reported by Spark, and samples are drawn with stratification over these bins. This ensures that each calibration run contains a representative mix of stages across the full range of input data volumes. The NNLS calibration is repeated on each resampled dataset, yielding an empirical distribution for each coefficient.

The resulting estimates, summarized in Table 9 and illustrated in Figure 6, exhibit low variance and narrow 95% confidence intervals, indicating strong parameter stability across resampling iterations.

TABLE 9. Structural coefficient confidence intervals.

Coefficient	Mean $\pm$ Std. Dev.	95% Confidence Interval	
		2.5%	97.5%
β_{bw}	1.0029 ± 0.0034	0.9960	1.0091
$\beta_{dynamic}$	2.0813 ± 0.0843	1.9215	2.2594
$\beta_{storage}$	13.4954 ± 0.1824	13.1671	13.8530

The physical interpretation of these parameters is as follows:

- $\beta_{bw} \approx 1.0$: This coefficient scales the bandwidth-dependent transfer term. Its proximity to 1.0 suggests that the analytical throughput formulation provides a good approximation of bulk data transfer behavior in our setup.
- $\beta_{dynamic} \approx 2.08$: This coefficient calibrates the additional overhead during the congestion-control ramp-up phase (e.g., slow-start). A value greater than 1 indicates that real transport behavior deviates from the idealized growth model and includes implementation-specific effects (e.g., early exit mechanisms) and protocol overheads that are not explicitly modeled.

- $\beta_{storage} \approx 13.50$: This comparatively large coefficient quantifies the contribution of the per-partition access latency term described in Section III-E. It reflects the repeated costs of remote data access, including multiple round-trips for connection establishment, authentication, and metadata retrieval operations in columnar storage formats.

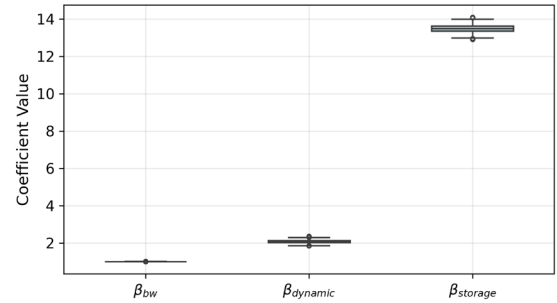


FIGURE 6. Stability of structural coefficients across 1,000 bootstrap iterations.

Two of the three model components depend on round-trip latency: $T_{storage}^*$ through RTT-bound HDFS partition-access RPCs, and $T_{dynamic}^*$ through the duration of the TCP slow-start phase. This shared dependence could imply larger dispersion in $\beta_{dynamic}$ and $\beta_{storage}$ relative to β_{bw} when the calibration set changes. To quantify the resulting collinearity structure, we computed the Variance Inflation Factor (VIF) [71] for each predictor on the full predictor matrix (36,821 stage observations $\times$ 3 predictors). It is defined as $VIF_j = (1 - R_j^2)^{-1}$, where j indexes each of the three predictors and R_j^2 is the proportion of variance in T_j^* explained by the other two. VIF is a widely used diagnostic for predictor collinearity; values above 10 are commonly cited as a threshold for severe collinearity [71]. The resulting values of 1.16, 2.53, and 2.78 for T_{bw}^* , $T_{dynamic}^*$, and $T_{storage}^*$ respectively indicate moderate collinearity, well below the threshold of 10.

To further assess robustness, we performed identification separately on Parquet-only and ORC-only subsets. The β_{bw} and $\beta_{dynamic}$ estimates remained within overlapping 95% confidence intervals across formats, indicating stability of the transport-layer terms within statistical uncertainty. For $\beta_{storage}$, the mean estimate was 14.01 for ORC and 13.11 for Parquet, with overlapping confidence intervals. This moderate variation is consistent with the structural role of $\beta_{storage}$, which captures aggregated per-partition access costs influenced by format-specific metadata organization. Because metadata access behavior is not explicitly parameterized but absorbed into $\beta_{storage}$, some format-dependent variation of this coefficient is expected and does not contradict the structural validity of the model.

We acknowledge that all β coefficients implicitly capture infrastructure-specific characteristics not explicitly modeled in (6), (8), and (12). Changes in system parameters, such

as packet loss, hardware architecture, HDFS configuration, storage backend, or security protocol behavior, may alter effective transport dynamics. In such cases, the system identification procedure should be repeated to recalibrate the coefficients.

2) CROSS-WORKLOAD STABILITY

The bootstrap procedure of Section VII-C1 resamples stages from the full pool, preserving the workload composition of the calibration set. To assess whether the identified coefficients depend on which query templates are included, we performed a workload-level cross-validation. The 99 TPC-DS templates were randomly split into an 80-template training set and a 19-template held-out set, NNLS calibration was run on the training stages, and the procedure was repeated for 1,000 independent partitions.

The resulting coefficient estimates are summarized in Table 10. The means closely match the full-pool fit reported in Table 9, indicating that random partitioning does not introduce a systematic bias. The observed dispersion pattern is consistent with the collinearity structure characterized in Section VII-C1.

TABLE 10. Structural coefficient stability under 1,000-iteration workload-level cross-validation (80/19 split of TPC-DS query templates).

Coefficient	Mean $\pm$ Std. Dev.	95% Confidence Interval	
		2.5%	97.5%
β_{bw}	1.0030 ± 0.0070	0.9881	1.0156
$\beta_{dynamic}$	2.0850 ± 0.1500	1.7472	2.3573
$\beta_{storage}$	13.4869 ± 0.5029	12.4391	14.4905

3) CROSS-ENVIRONMENT GENERALIZATION

To evaluate the proposed model's ability to generalize across different network regimes, we performed a restricted calibration experiment. The system identification procedure was executed exclusively using measurements collected at 10 Gb/s and 10.49 ms RTT. No samples from other bandwidth configurations or other latency conditions were included in the calibration set. Under this restricted calibration, the estimated coefficients remained structurally consistent with the full-dataset identification. The bandwidth and storage coefficients retained comparable magnitudes, while the dynamic transport coefficient exhibited substantially higher variance, as shown in Table 11.

TABLE 11. Structural coefficient confidence intervals under restricted calibration (calibrated at 10 Gb/s, 10.49 ms).

Coefficient	Mean $\pm$ Std. Dev.	95% Confidence Interval	
		2.5%	97.5%
β_{bw}	1.0130 ± 0.0075	0.9986	1.0287
$\beta_{dynamic}$	1.5703 ± 0.4718	0.5900	2.4672
$\beta_{storage}$	13.8017 ± 0.3900	13.0380	14.5763

Despite this increased uncertainty, stage-level prediction accuracy remained high across all out-of-distribution

configurations outside the calibration set, as shown in Table 12. R^2 exceeded 0.95 across all evaluated configurations. Even in the most network-constrained evaluated setting (2 Gb/s, 25.46 ms), the model achieved $R^2 = 0.9759$ with low residual bias, indicating extrapolation capability across a $5\times$ reduction in bandwidth and a $2.5\times$ increase in RTT relative to the calibration conditions.

TABLE 12. Out-of-distribution stage-level prediction accuracy (calibrated at 10 Gb/s, 10.49 ms).

Bandwidth	Latency	MAE	RMSE	WMAPE	R^2
25 Gb/s	0.30 ms	0.1915	0.7674	7.39%	0.9893
25 Gb/s	10.49 ms	0.3581	0.9180	9.84%	0.9875
25 Gb/s	25.46 ms	0.8947	2.3259	15.15%	0.9621
10 Gb/s	0.30 ms	0.1840	0.6163	6.87%	0.9933
10 Gb/s	10.49 ms	0.3625	0.9116	9.61%	0.9890
10 Gb/s	25.46 ms	0.9373	2.4935	15.55%	0.9582
5 Gb/s	0.30 ms	0.2180	0.5939	6.56%	0.9956
5 Gb/s	10.49 ms	0.4374	1.0768	10.41%	0.9878
5 Gb/s	25.46 ms	0.9674	2.3451	15.37%	0.9667
2 Gb/s	0.30 ms	0.5749	1.4449	9.25%	0.9927
2 Gb/s	25.46 ms	1.2752	2.8994	15.48%	0.9759

These results suggest that the model retains high predictive accuracy when calibrated on a single network configuration and evaluated under substantially different bandwidth and latency conditions. However, the results also show that reliable system identification depends on the diversity of network conditions included during calibration. When we restricted calibration to low-latency conditions (0.30 ms), latency-related effects were only weakly reflected in the data. In such cases, coefficients associated with latency-driven transport components, particularly $\beta_{dynamic}$, became more difficult to identify reliably. Therefore, for robust deployment, calibration should be performed under network conditions that are representative of the expected operating range.

4) ONE-AT-A-TIME COEFFICIENT SENSITIVITY

To assess the impact of coefficient miscalibration on stage-level prediction quality, we apply a one-at-a-time sensitivity analysis [72]. Each coefficient is set independently to the 95% confidence interval limits observed in the restricted single-configuration calibration in Section VII-C3, with results reported in Table 13. In each sensitivity row, one coefficient is set to a confidence interval bound while the other two remain at the restricted mean. For example, the $\beta_{dynamic} = 0.5900$ row uses $\beta_{bw} = 1.0130$ and $\beta_{storage} = 13.8017$. The restricted mean row sets all three coefficients to their mean values from Table 11, and the transport-agnostic predictor serves as the comparison baseline. Results are shown at 2 Gb/s and 25 Gb/s, both at 25.46 ms RTT. In every case the model outperforms the transport-agnostic predictor and remains close to the calibrated baseline.

5) SENSITIVITY TO MTU AND CONGESTION-WINDOW SCALING

To evaluate the model's sensitivity to link-layer parameters, we compare predictive performance under two Maximum

TABLE 13. Stage-level prediction accuracy under one-at-a-time coefficient sensitivity at the 95% confidence interval limits of the restricted calibration.

Case	MAE	RMSE	WMAPE	R^2
2 Gb/s, 25.46 ms RTT				
Transport-agnostic	5.67	14.98	68.8%	0.3555
Restricted mean	1.28	2.90	15.5%	0.9759
$\beta_{bw} = 0.9986$	1.25	2.78	15.2%	0.9779
$\beta_{bw} = 1.0287$	1.31	3.05	15.9%	0.9733
$\beta_{dynamic} = 0.5900$	1.25	2.83	15.2%	0.9771
$\beta_{dynamic} = 2.4672$	1.30	2.98	15.8%	0.9745
$\beta_{storage} = 13.0380$	1.26	2.82	15.3%	0.9772
$\beta_{storage} = 14.5763$	1.30	3.01	15.8%	0.9741
25 Gb/s, 25.46 ms RTT				
Transport-agnostic	3.33	7.43	56.4%	0.6132
Restricted mean	0.89	2.33	15.2%	0.9621
$\beta_{bw} = 0.9986$	0.91	2.36	15.5%	0.9610
$\beta_{bw} = 1.0287$	0.88	2.30	14.8%	0.9630
$\beta_{dynamic} = 0.5900$	1.05	2.69	17.8%	0.9494
$\beta_{dynamic} = 2.4672$	0.83	2.19	14.1%	0.9664
$\beta_{storage} = 13.0380$	0.95	2.43	16.1%	0.9586
$\beta_{storage} = 14.5763$	0.85	2.25	14.4%	0.9644

Transmission Unit (MTU) settings: standard Ethernet frames (MTU = 1500) and jumbo frames (MTU = 9001). These values correspond to supported MTU configurations in hybrid interconnect solutions, making the evaluation representative of practical deployments [73].

In modern TCP implementations, the initial congestion window depends on the negotiated path MTU and is influenced by link-layer configuration [74]. Consequently, MTU directly affects the effective initial congestion window size. Through (12), MTU impacts the transient ramp-up cost required to reach the bandwidth-delay product. We evaluate this sensitivity under 25 Gb/s and 25.46 ms RTT, where slow-start and bandwidth-delay product effects are most pronounced. Reducing MTU from 9001 to 1500 decreases the predictive performance of the transport-agnostic baseline from $R^2 = 0.6132$ to $R^2 = 0.5538$, indicating increased unexplained variance under slower ramp-up conditions. A smaller MTU reduces the effective initial congestion window, thereby increasing the number of round-trip intervals required to reach W_{target} in (12).

Importantly, no recalibration of the β coefficients was performed for the MTU variation. Using the same calibrated parameters, the transport-aware model maintains stable predictive performance, with $R^2 = 0.9621$ for MTU = 9001 and $R^2 = 0.9595$ for MTU = 1500. Overall, the MTU experiment demonstrates that the proposed formulation maintains predictive accuracy under realistic link-layer variations.

It should be noted that the exact initial congestion window and slow-start behavior depend on the specific TCP implementation and system configuration [75]. In our formulation, W_{init} serves as a structural approximation.

D. JOB-LEVEL RUNTIME PREDICTION

We now assess end-to-end job runtime prediction accuracy across all evaluated bandwidth-latency configurations.

The proposed transport-aware model, combined with the dependency-aware aggregation described in Section III-G, is compared against a transport-agnostic baseline. This baseline estimates job runtime as the cumulative sum of stage durations, following the aggregation approach used in prior work [14]. Table 14 summarizes prediction accuracy across MAE, RMSE, WMAPE, and R^2 evaluation metrics for 495 TPC-DS query executions.

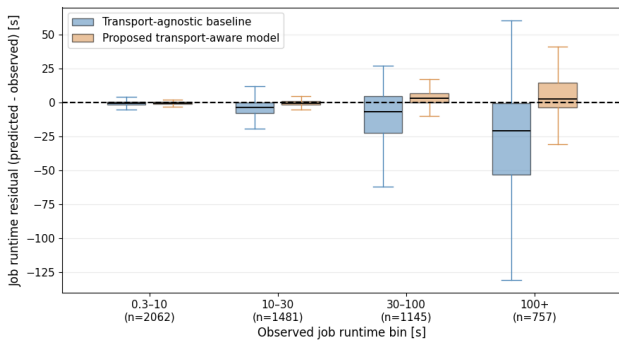
Across all evaluated configurations, the proposed transport-aware model with dependency-aware aggregation consistently outperforms the transport-agnostic cumulative baseline in hybrid cloud scenarios. Under low-latency and high-bandwidth conditions, both approaches achieve high accuracy. Here, the network effects are limited, and the primary difference stems from the aggregation strategy. By accounting for stage parallelism and resource constraints rather than relying on simple cumulative summation, the proposed model reduces MAE by approximately 33.17% even when stage-level prediction differences are small. At 25.46 ms RTT and 25 Gb/s bandwidth, MAE decreases from 15.98 s in the cumulative baseline to 4.69 s with the proposed formulation, while R^2 improves from 0.8975 to 0.9916. The RMSE and WMAPE columns in Table 14 confirm this pattern; in high-latency regimes, the proposed model yields substantially lower prediction error than the baseline. In this regime, both improved stage modeling and dependency-aware aggregation contribute to the observed gains. The largest differences appear in bandwidth-constrained scenarios. Under the most challenging configuration (2 Gb/s, 25.46 ms), MAE is reduced from 28.20 s to 6.54 s, representing a 76.8% reduction in absolute error, while R^2 improves from 0.6625 to 0.9858. Even under low-latency conditions at 2 Gb/s, MAE decreases from 19.37 s to 2.22 s, representing an 88.5% reduction.

Figure 7 illustrates the distribution of job runtime residuals (predicted runtime minus observed runtime), where the dashed horizontal line at 0 denotes perfect prediction. The residuals are grouped into four interpretable bins (0.3-10 s, 10-30 s, 30-100 s, and 100+ s), with the sample counts (n) for each category indicated on the x-axis. Compared to the transport-agnostic baseline, the proposed model produces residuals that are more tightly concentrated around zero across all runtime bins, in contrast to the baseline's pronounced under-prediction tendency for long-running jobs.

The overall prediction characteristics are further exemplified in Figure 8, which provides a side-by-side comparison of the transport-agnostic baseline (blue) and the proposed transport-aware model (orange) across all evaluated hybrid configurations, consolidating 5,445 data points representing 495 distinct query templates evaluated across each of the eleven network regimes. The diagonal line in the figure represents perfect prediction. While the baseline shows significant scattering, the proposed model's data points are distributed close to the diagonal line, indicating a strong agreement between the predicted and observed values. This alignment is maintained across a wide range of execution

TABLE 14. Job-level runtime prediction accuracy under varying bandwidth and latency conditions.

Bandwidth	Latency	MAE (s)		RMSE		WMAPE (%)		R ²	
		Baseline	Proposed	Baseline	Proposed	Baseline	Proposed	Baseline	Proposed
25 Gb/s	0.30 ms	4.40	2.94	9.50	6.58	11.75	7.86	0.9851	0.9928
25 Gb/s	10.49 ms	6.15	3.26	11.07	6.55	14.23	7.53	0.9809	0.9933
25 Gb/s	25.46 ms	15.98	4.69	28.22	8.09	28.99	8.50	0.8975	0.9916
10 Gb/s	0.30 ms	4.56	2.81	9.98	6.14	12.06	7.44	0.9833	0.9937
10 Gb/s	10.49 ms	6.58	3.34	11.96	6.58	15.04	7.63	0.9777	0.9933
10 Gb/s	25.46 ms	16.62	4.94	29.18	8.62	29.78	8.85	0.8917	0.9906
5 Gb/s	0.30 ms	6.77	2.67	16.56	5.97	16.23	6.40	0.9583	0.9946
5 Gb/s	10.49 ms	8.74	3.57	18.42	6.95	18.97	7.76	0.9492	0.9928
5 Gb/s	25.46 ms	17.90	5.17	33.04	9.65	31.28	9.03	0.8628	0.9883
2 Gb/s	0.30 ms	19.37	2.22	53.93	4.63	33.58	3.85	0.7127	0.9979
2 Gb/s	25.46 ms	28.20	6.54	61.90	12.71	41.25	9.57	0.6625	0.9858


FIGURE 7. Job-level prediction residuals by runtime bin: baseline (blue) vs. proposed (orange) model.

times, spanning from 0.46 s to 659.46 s. Importantly, the proposed model maintains this accuracy despite the highly heterogeneous degradation patterns discussed in Section VI, where some workloads exhibited negligible slowdown while others experienced degradation exceeding 1000%. Even under constrained bandwidth-delay conditions, the dispersion remains limited, and no systematic overestimation or underestimation is observed. Larger deviations occur primarily for short-duration jobs, where fixed overheads and measurement noise account for a larger fraction of total runtime. Overall, the close alignment with the diagonal supports the high R^2 values reported in Table 14 and demonstrates that the proposed transport-aware formulation generalizes reliably across heterogeneous network environments.

VIII. DISCUSSION

The experimental results highlight several broader implications for hybrid cloud performance modeling. Network transport must be treated as a primary performance factor in hybrid environments. When bandwidth is abundant and latency is low, transport-agnostic predictors can remain reasonably accurate. However, once the bandwidth-delay product or throughput saturation dominates, prediction error increases rapidly. These results indicate that hybrid-cloud performance cannot be reliably modeled under heterogeneous network conditions without accounting for transport-layer behavior. The current framework characterizes the hybrid interconnect

through steady-state parameters: available bandwidth and round-trip delay. We do not explicitly model transient effects such as packet loss, congestion spikes, or jitter. In a real-world setting, packet loss triggers TCP retransmissions and congestion window reductions, which effectively reduce throughput below the model's steady-state assumptions. Similarly, congestion spikes create transient throughput drops not captured by the current static bandwidth parameter, and jitter introduces latency variability that primarily affects stages with many small HDFS partition accesses. Micro-bursting can create short-lived throughput variations that the static bandwidth parameter does not capture, and shared WAN links introduce competing traffic that may reduce available bandwidth below the provisioned rate. In a multi-tenant environment, if the effective bandwidth at submission time is measured rather than assumed equal to the provisioned rate, the model already reflects the current contention level through its B_{link} input parameter. The remaining limitation is bandwidth variability during execution, where contention changes after the prediction is issued. Incorporating explicit variables for these transient behaviors remains an objective for future work.

Stage-level modeling was shown to be an effective abstraction for prediction in the evaluated setting. Dividing execution into homogeneous task groups separated by shuffle boundaries provides a natural structural unit for analysis. The consistently high stage-level accuracy across different bandwidth and latency settings suggests that this granularity captures the dominant performance drivers while remaining analytically manageable.

Accurate stage-level modeling alone is not sufficient for accurate job-level prediction. Even moderate stage-level errors can accumulate when simple cumulative aggregation is used. Ignoring parallel execution and the critical path leads to systematic overestimation in our experiments. The results indicate that combining transport-aware stage prediction with dependency-aware aggregation is necessary to achieve consistently accurate end-to-end estimates under the evaluated hybrid conditions.

The stability of the calibrated coefficients across storage formats and compression strategies suggests that the model captures general transport behavior rather than

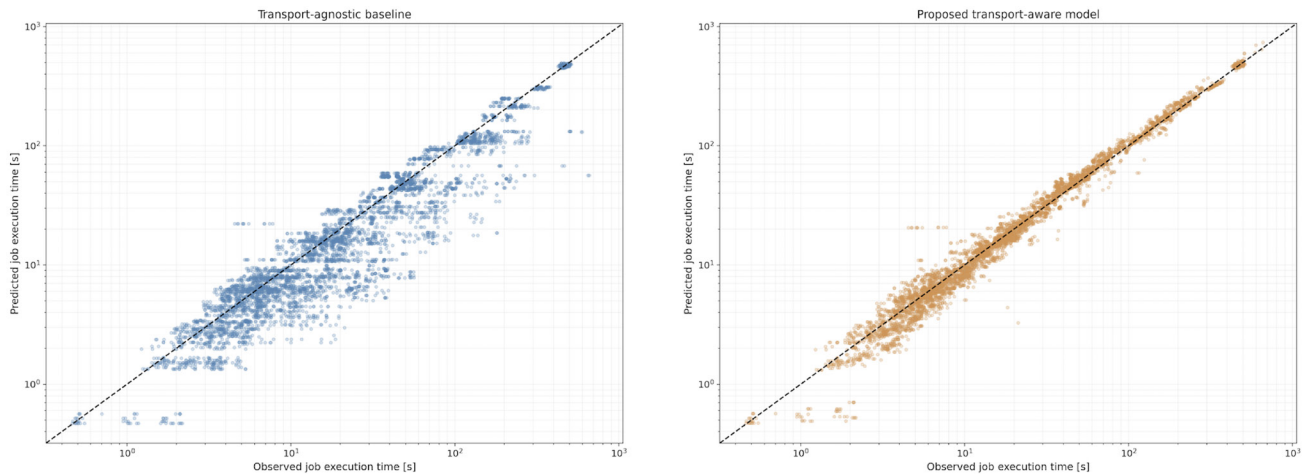


FIGURE 8. Predicted vs. observed execution times: baseline (blue) and proposed (orange) models across all network regimes.

workload-specific artifacts within the studied benchmark. At the same time, the restricted-calibration experiment demonstrates that reliable extrapolation depends on sufficiently diverse calibration data. Calibration should therefore cover representative bandwidth and latency values expected in production deployments.

The current scope covers recurring workloads with available on-premises execution profiles. Ad hoc queries without prior telemetry are explicitly out of scope (Section III-A). Streaming workloads are similarly outside scope: fluctuating ingestion rates and variable data volumes per micro-batch prevent the construction of stable stage-level execution signatures from historical data. Within this scope, workload instability across successive executions of a recurring template, such as changes in input data volume or partition skew, can shift the execution signature away from the last profiled run. Because the stored profile is refreshed with each on-premises execution, gradual evolution is tracked automatically. When drift becomes significant, profile age or prediction residuals can flag the condition (Section IV-B). To further extend coverage to highly dynamic or unseen workloads, one direction is static execution plan analysis: the physical plan of a new query exposes partition counts, scan sizes, and stage boundaries that can serve directly as model inputs without prior execution. Complementary to this, incremental transfer learning from historical workload knowledge [36] could further improve prediction accuracy for previously unseen job types by leveraging similarities to known workloads. Extending the framework to cloud-native object stores would require adapting the storage-access latency model and the TCP connection mapping assumption to reflect object-store access semantics, and recalibrating the structural coefficients against the target infrastructure. Specifically, three structural differences require adaptation: persistent HTTPS connections reduce the effective independent flow count below the active task count, request multiplexing over shared connections alters the per-partition

latency accounting in Section III-E, and per-object metadata overhead introduces additional latency costs not present in HDFS block reads.

Extending validation to real-world production workloads is constrained by data privacy and security policies that typically restrict access to execution telemetry, making controlled, reproducible experiments infeasible with such data. The TPC-DS benchmark models the query diversity and data skew of retail data warehouse workloads at production scale and is therefore used as a representative proxy. To support external validation, the complete dataset and reference implementation will be released alongside the paper.

The proposed approach represents a middle ground between purely analytical and purely data-driven methods, consistent with the Physics-Informed Machine Learning (PIML) perspective introduced in Section II-C. The framework adopts an inductive bias strategy in two ways: through analytically derived, physically grounded transport terms that approximate key network-side mechanisms, and through the use of NNLS for system identification. The non-negativity constraint ($\beta \geq 0$) further restricts the hypothesis space to physically valid regions and supports interpretable solutions. In this way, the framework preserves physical interpretability and extrapolation capability while avoiding per-query pilot runs. The results indicate that the proposed system identification method can provide both generalization and operational practicality for hybrid cloud performance prediction under heterogeneous network conditions, supporting offloading decisions at the moment of job submission.

IX. CONCLUSION

This paper presents a transport-aware performance modeling framework to support submission-time decision-making for hybrid cloud bursting in network-constrained environments. The approach decouples transport effects from application logic by reusing historical on-premises execution profiles and

projecting them to hybrid settings through a small set of physically grounded transport terms.

Across TPC-DS workloads at 1 TB scale, under controlled network emulation covering diverse bandwidth, latency, storage format, and compression settings, the proposed framework maintained high predictive accuracy and demonstrated generalization beyond the specific calibration setting. At the job level, combining transport-aware stage prediction with dependency-aware aggregation substantially improves accuracy over a transport-agnostic cumulative baseline. Under the most challenging evaluated configuration, the proposed framework reduces the MAE of execution-time predictions by 76.8% (from 28.20 s to 6.54 s), while increasing R^2 from 0.6625 to 0.9858.

Several extensions can broaden the scope of the proposed framework. Evaluating the model under more adverse network conditions, including controlled packet loss and latency variability, would further assess robustness beyond steady-state bandwidth and RTT regimes. Further directions include adapting the framework to cloud-native object stores, enabling online recalibration of structural coefficients from production telemetry, extending coverage to ad hoc workloads without prior on-premises profiles, and embedding the predictor into cost-aware placement policies.

ACKNOWLEDGMENT

The authors used Gemini 3 Flash, a large language model developed by Google, to improve the language and readability of the manuscript. They are solely responsible for the content and conclusions of this work.

ABBREVIATIONS

Abbreviation	Definition
AQE	Adaptive Query Execution.
AWS	Amazon Web Services.
CDP	Cloudera Data Platform.
DAG	Directed Acyclic Graph.
ET _r	Execution Time Degradation.
HDFS	Hadoop Distributed File System.
HTB	Hierarchical Token Bucket.
JVM	Java Virtual Machine.
MAE	Mean Absolute Error.
MAPE	Mean Absolute Percentage Error.
MIP	Mixed-Integer Programming.
ML	Machine Learning.
MTU	Maximum Transmission Unit.
NNLS	Non-Negative Least Squares.
PIML	Physics-Informed Machine Learning.
PINNs	Physics-Informed Neural Networks.
RDD	Resilient Distributed Dataset.
REST	Representational State Transfer.
RMSE	Root Mean Square Error.
RPC	Remote Procedure Call.
RTT	Round-Trip Time.
SAN	Stochastic Activity Network.

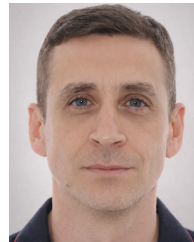
SASL	Simple Authentication and Security Layer.
TCP	Transmission Control Protocol.
TLS	Transport Layer Security.
TPC-DS	Transaction Processing Performance Council Decision Support.
TSG	Traffic Shaping Gateway.
VIF	Variance Inflation Factor.
WAN	Wide Area Network.
WMAPE	Weighted Mean Absolute Percentage Error.
YARN	Yet Another Resource Negotiator.

REFERENCES

- [1] I. Zablach, S. Diaz, C. Agudelo-Santos, Y. Molina, M. Madrid, A. Diaz, and M. Madrid, "Economic viability of outsourcing cluster computing to the cloud: Insights for technological entrepreneurship and regional innovation in Honduras," in *Proc. LACCEI*, Dec. 2025, pp. 1–6, doi: [10.18687/leird2025.1.1.889](https://doi.org/10.18687/leird2025.1.1.889).
- [2] Amazon Web Services. *AWS Post-Event Summaries (PES)*. Accessed: Apr. 12, 2026. [Online]. Available: <https://aws.amazon.com/premiumsupport/technology/pes/>
- [3] Microsoft Azure. *Azure Post Incident Reviews (PIRs)*. Accessed: Apr. 12, 2026. [Online]. Available: <https://azure.status.microsoft.com/en-us/status/history/>
- [4] A. Alsadah and H. Alhajjaj, "Challenges of cloud solutions adoption in large corporations," *EAI Endorsed Trans. Cloud Syst.*, vol. 5, no. 14, Mar. 2019, Art. no. 158415, doi: [10.4108/eai.13-7-2018.158415](https://doi.org/10.4108/eai.13-7-2018.158415).
- [5] Z. Qiu, H. Park, J. Zhao, Y.-K. Wang, A. Balyan, G. Singh, Y. Zhang, S. J. Song, G. R. Ganger, and G. Amvrosiadis, "Moirai: Optimizing placement of data and compute in hybrid clouds," in *Proc. ACM SIGOPS 31st Symp. Operating Syst. Princ. (SOSP)*, Oct. 2025, pp. 875–891.
- [6] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of MapReduce workloads," *Proc. VLDB Endowment*, vol. 5, no. 12, pp. 1802–1813, Aug. 2012, doi: [10.14778/2367502.2367519](https://doi.org/10.14778/2367502.2367519).
- [7] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016, doi: [10.1145/2934664](https://doi.org/10.1145/2934664).
- [8] M. Kornacker et al., "Impala: A modern, open-source SQL engine for Hadoop," in *Proc. 7th Biennial Conf. Innov. Data Syst. Res. (CIDR)*, Jan. 2015. [Online]. Available: https://www.cidrdb.org/cidr2015/Papers/CIDR15_Paper28.pdf
- [9] J. Camacho-Rodríguez, A. Chauhan, A. Gates, E. Koifman, O. O'Malley, V. Garg, Z. Haindrich, S. Shelukhin, P. Jayachandran, S. Seth, D. Jaiswal, S. Bouguerra, N. Bangarwa, S. Hariappan, A. Agarwal, J. Dere, D. Dai, T. Nair, N. Dembla, G. Vijayaraghavan, and G. Hagleitner, "Apache hive: From MapReduce to enterprise-grade big data warehousing," in *Proc. Int. Conf. Manage. Data (SIGMOD)*, Jun. 2019, pp. 1773–1786.
- [10] R. Sethi, M. Traverso, D. Sundstrom, D. Phillips, W. Xie, Y. Sun, N. Yegitbasi, H. Jin, E. Hwang, N. Shingte, and C. Berner, "Presto: SQL on everything," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 1802–1813, doi: [10.1109/ICDE.2019.00196](https://doi.org/10.1109/ICDE.2019.00196).
- [11] A. Kanto and A. C. Marosi, "Optimizing big data workloads in hybrid cloud environments: Evaluating file formats and compression codecs," in *Proc. IEEE 23rd Jubilee Int. Symp. Intell. Syst. Informat. (SISY)*, Sep. 2025, pp. 31–36, doi: [10.1109/SISY67000.2025.11205406](https://doi.org/10.1109/SISY67000.2025.11205406).
- [12] S. Venkataraman, Z. Yang, M. J. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient performance prediction for large-scale advanced analytics," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Mar. 2016, pp. 363–378.
- [13] S. Karimian-Aliabadi, D. Ardagna, R. Entezari-Maleki, E. Gianniti, and A. Movaghar, "Analytical composite performance models for big data applications," *J. Netw. Comput. Appl.*, vol. 142, pp. 63–75, Sep. 2019, doi: [10.1016/j.jnca.2019.06.009](https://doi.org/10.1016/j.jnca.2019.06.009).
- [14] S. Mustafa, I. Elghandour, and M. A. Ismail, "A machine learning approach for predicting execution time of spark jobs," *Alexandria Eng. J.*, vol. 57, no. 4, pp. 3767–3778, Dec. 2018, doi: [10.1016/j.aej.2018.03.006](https://doi.org/10.1016/j.aej.2018.03.006).

- [15] S. Shah, Y. Amannejad, D. Krishnamurthy, and M. Wang, "Quick execution time predictions for spark applications," in *Proc. 15th Int. Conf. Netw. Service Manage. (CNSM)*, Oct. 2019, pp. 1–9, doi: [10.23919/CNSM46954.2019.9012752](https://doi.org/10.23919/CNSM46954.2019.9012752).
- [16] N. Ahmed, A. L. C. Barczak, M. A. Rashid, and T. Susnjak, "Runtime prediction of big data jobs: Performance comparison of machine learning algorithms and analytical models," *J. Big Data*, vol. 9, no. 1, p. 67, May 2022, doi: [10.1186/s40537-022-00623-1](https://doi.org/10.1186/s40537-022-00623-1).
- [17] H. Tariq and O. Das, "A deterministic model to predict execution time of Spark applications," in *Proc. Comput. Perform. Eng.*, vol. 13659, Cham, Switzerland: Springer, 2023, pp. 167–181.
- [18] K. U. Farooq, H. A. Sanjay, and G. C. Akshatha, "Predicting big data applications runtime on spark cluster using machine learning techniques," in *Proc. IEEE Int. Conf. Electron., Comput. Commun. Technol. (CONECT)*, Jul. 2025, pp. 1–6, doi: [10.1109/CONECT65861.2025.11306724](https://doi.org/10.1109/CONECT65861.2025.11306724).
- [19] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST)*, May 2010, pp. 1–10, doi: [10.1109/MSST.2010.5496972](https://doi.org/10.1109/MSST.2010.5496972).
- [20] Amazon Web Services. *Amazon Simple Storage Service (Amazon S3)*. Accessed: Apr. 12, 2026. [Online]. Available: <https://aws.amazon.com/s3/>
- [21] Microsoft Azure. *Azure Data Lake Storage*. Accessed: Apr. 12, 2026. [Online]. Available: <https://azure.microsoft.com/en-us/products/storage/data-lake-storage>
- [22] A. Kanto. *Transport-Aware Performance Modeling Reference Implementation*. Accessed: Apr. 12, 2026. [Online]. Available: <https://github.com/cloudera/transport-aware-spark-model>
- [23] D. Loreti and A. Ciampolini, "A hybrid cloud infrastructure for big data applications," in *Proc. IEEE 17th Int. Conf. High Perform. Comput. Commun. 7th Int. Symp. Cyberspace Saf. Secur., IEEE 12th Int. Conf. Embedded Softw. Syst.*, Aug. 2015, pp. 1713–1718, doi: [10.1109/HPCC-CSS-ICSS.2015.140](https://doi.org/10.1109/HPCC-CSS-ICSS.2015.140).
- [24] F. J. Clemente-Castelló, B. Nicolae, R. Mayo, and J. C. Fernández, "Performance model of MapReduce iterative applications for hybrid cloud bursting," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 8, pp. 1794–1807, Aug. 2018, doi: [10.1109/TPDS.2018.2802932](https://doi.org/10.1109/TPDS.2018.2802932).
- [25] F. Ullah, S. Dhingra, X. Xia, and M. A. Babar, "Evaluation of distributed data processing frameworks in hybrid clouds," *J. Netw. Comput. Appl.*, vol. 224, Apr. 2024, Art. no. 103837, doi: [10.1016/j.jnca.2024.103837](https://doi.org/10.1016/j.jnca.2024.103837).
- [26] K. Y. Rajput, L. Xiaoping, and A. Lakhan, "Spark workflow task scheduling with deadline and privacy constraints in hybrid cloud networks," *Soft Comput.*, vol. 29, no. 2, pp. 783–801, Feb. 2025, doi: [10.1007/s00500-025-10486-2](https://doi.org/10.1007/s00500-025-10486-2).
- [27] Y. Huang, Y. Shi, Z. Zhong, Y. Feng, J. Cheng, J. Li, H. Fan, C. Li, T. Guan, and J. Zhou, "Yugong: Geo-distributed data and job placement at scale," *Proc. VLDB Endowment*, vol. 12, no. 12, pp. 2155–2169, Aug. 2019, doi: [10.14778/3352063.3352132](https://doi.org/10.14778/3352063.3352132).
- [28] R.-I. Roman, B. Nicolae, A. Costan, and G. Antoniu, "Understanding spark performance in hybrid and multi-site clouds," in *Proc. 6th Int. Workshop Big Data Analyt., Challenges Opportunities (BDAC)*, Austin, TX, USA, Nov. 2015. [Online]. Available: <https://inria.hal.science/hal-01239140>
- [29] OpenStack Foundation. *OpenStack*. Accessed: Apr. 12, 2026. [Online]. Available: <https://www.openstack.org/>
- [30] Microsoft. *Microsoft Azure*. Accessed: Apr. 12, 2026. [Online]. Available: <https://azure.microsoft.com>
- [31] V. P. Verma, S. Kumar, and N. S. Naik, "Adaptive node-oriented data placement for heterogeneous Hadoop clusters," *IEEE Access*, vol. 13, pp. 79548–79559, 2025, doi: [10.1109/ACCESS.2025.3565759](https://doi.org/10.1109/ACCESS.2025.3565759).
- [32] E. M. Mendes and J. R. De Almeida, "Hybrid data lake entities localization optimization based on complex networks," *IEEE Access*, vol. 14, pp. 12744–12749, 2026, doi: [10.1109/ACCESS.2026.3656719](https://doi.org/10.1109/ACCESS.2026.3656719).
- [33] G. Cheng, S. Ying, B. Wang, and Y. Li, "Efficient performance prediction for apache spark," *J. Parallel Distrib. Comput.*, vol. 149, pp. 40–51, Mar. 2021, doi: [10.1016/j.jpdc.2020.10.010](https://doi.org/10.1016/j.jpdc.2020.10.010).
- [34] S. Karimian-Aliabadi, M.-M. Aseman-Manzar, R. Entezari-Maleki, D. Ardagna, B. Egger, and A. Movaghar, "Fixed-point iteration approach to spark scalable performance modeling and evaluation," *IEEE Trans. Cloud Comput.*, vol. 11, no. 1, pp. 897–910, Jan. 2023, doi: [10.1109/TCC.2021.3119943](https://doi.org/10.1109/TCC.2021.3119943).
- [35] C. Shen, C. Chen, and G. Rao, "A novel multi-task performance prediction model for spark," *Appl. Sci.*, vol. 13, no. 22, p. 12242, Nov. 2023, doi: [10.3390/app132212242](https://doi.org/10.3390/app132212242).
- [36] M. Garraalda-Barrio, C. Eiras-Franco, and V. Bolón-Canedo, "Adaptive incremental transfer learning for efficient performance modeling of big data workloads," *Future Gener. Comput. Syst.*, vol. 166, May 2025, Art. no. 107730, doi: [10.1016/j.future.2025.107730](https://doi.org/10.1016/j.future.2025.107730).
- [37] Z. Chao, S. Shi, H. Gao, J. Luo, and H. Wang, "A gray-box performance model for apache spark," *Future Gener. Comput. Syst.*, vol. 89, pp. 58–67, Dec. 2018, doi: [10.1016/j.future.2018.06.032](https://doi.org/10.1016/j.future.2018.06.032).
- [38] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, pp. 686–707, Feb. 2019, doi: [10.1016/j.jcp.2018.10.045](https://doi.org/10.1016/j.jcp.2018.10.045).
- [39] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Rev. Phys.*, vol. 3, no. 6, pp. 422–440, May 2021, doi: [10.1038/s42254-021-00314-5](https://doi.org/10.1038/s42254-021-00314-5).
- [40] K. Luo, J. Zhao, Y. Wang, J. Li, J. Wen, J. Liang, H. Soekmadji, and S. Liao, "Physics-informed neural networks for PDE problems: A comprehensive review," *Artif. Intell. Rev.*, vol. 58, no. 10, p. 323, Jul. 2025, doi: [10.1007/s10462-025-11322-7](https://doi.org/10.1007/s10462-025-11322-7).
- [41] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific machine learning through physics-informed neural networks: Where we are and What's next," *J. Scientific Comput.*, vol. 92, no. 3, p. 88, Jul. 2022, doi: [10.1007/s10915-022-01939-z](https://doi.org/10.1007/s10915-022-01939-z).
- [42] S. A. Jyothi, C. Curino, I. Menache, S. Narayanamurthy, A. Tumanov, J. Yaniv, R. Mavlyutov, I. Goiri, S. Krishnan, J. Kulkarni, and S. Rao, "Morpheus: Towards automated SLOs for enterprise clusters," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, Nov. 2016, pp. 117–134.
- [43] A. Chung, S. Krishnan, K. Karanasos, C. Curino, and G. R. Ganger, "Unearthing inter-job dependencies for better cluster scheduling," in *Proc. 14th USENIX Conf. Operating Syst. Design Implement. (OSDI)*, Nov. 2020, pp. 1205–1223.
- [44] E. Begoli, J. Camacho-Rodríguez, J. Hyde, M. J. Mior, and D. Lemire, "Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources," in *Proc. Int. Conf. Manage. Data (SIGMOD)*, May 2018, pp. 221–230.
- [45] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia, "Spark SQL: Relational data processing in spark," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, May 2015, pp. 1383–1394.
- [46] M. Xue et al., "Adaptive and robust query execution for lakehouses at scale," *Proc. VLDB Endowment*, vol. 17, no. 12, pp. 3947–3959, Aug. 2024, doi: [10.14778/3685800.3685818](https://doi.org/10.14778/3685800.3685818).
- [47] I. Sfiligoi, M. Hare, D. Schultz, F. Würthwein, B. Riedel, T. Hutton, S. Barnet, and V. Brik, "Managing cloud networking costs for data-intensive applications by provisioning dedicated network links," in *Proc. Pract. Exper. Adv. Res. Comput. (PEARC)*, Jul. 2021, pp. 1–8, doi: [10.1145/3437359.3465563](https://doi.org/10.1145/3437359.3465563).
- [48] Apache Software Foundation. *Hadoop-AWS Module: Integration With Amazon Web Services*. Accessed: Apr. 15, 2026. [Online]. Available: <https://hadoop.apache.org/docs/current/hadoop-aws/tools/hadoop-aws/index.html>
- [49] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 3, pp. 67–82, Jul. 1997, doi: [10.1145/263932.264023](https://doi.org/10.1145/263932.264023).
- [50] X. Zeng, Y. Hui, J. Shen, A. Pavlo, W. McKinney, and H. Zhang, "An empirical evaluation of columnar storage formats," *Proc. VLDB Endowment*, vol. 17, no. 2, pp. 148–161, Oct. 2023, doi: [10.14778/3626292.3626298](https://doi.org/10.14778/3626292.3626298).
- [51] M. Allman, V. Paxson, and E. Blanton, *TCP Congestion Control*, Standard RFC 5681, Sep. 2009.
- [52] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008, doi: [10.1145/1400097.1400105](https://doi.org/10.1145/1400097.1400105).
- [53] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 303–314, Oct. 1998, doi: [10.1145/285243.285291](https://doi.org/10.1145/285243.285291).
- [54] W. Bao, V. W. S. Wong, and V. C. M. Leung, "A model for steady state throughput of TCP CUBIC," in *Proc. IEEE Global Telecommun. Conf. GLOBECOM*, Dec. 2010, pp. 1–6, doi: [10.1109/GLOBECOM.2010.5684172](https://doi.org/10.1109/GLOBECOM.2010.5684172).

- [55] P. Shankhdhar, F. Liu, J. Narale, J. Sun, R. Schluskel, and L. Antova, "Presto's history-based query optimizer," *Proc. VLDB Endowment*, vol. 17, no. 12, pp. 4077–4089, Aug. 2024, doi: [10.14778/3685800.3685828](https://doi.org/10.14778/3685800.3685828).
- [56] Apache Software Foundation. *Apache Airflow*. Accessed: Apr. 15, 2026. [Online]. Available: <https://airflow.apache.org/>
- [57] M. Luo, S. Zhuang, S. Vengadesan, R. Bhardwaj, J. Chang, E. Friedman, S. Shenker, and I. Stoica, "Starburst: A cost-aware scheduler for hybrid cloud," in *Proc. 2024 USENIX Annu. Tech. Conf. (USENIX ATC)*, Jul. 2024, pp. 37–57.
- [58] Cloudera. *The Shifting Airgapped Data Processing Market*. Accessed: Apr. 12, 2026. [Online]. Available: <https://www.cloudera.com/blog/business/the-shifting-airgapped-data-processing-market-what-it-means-for-the-public-sector.html>
- [59] Cloudera. *Cloudera Runtime Component Versions (CDP 7.3.1.100)*. Accessed: Apr. 12, 2026. [Online]. Available: <https://docs.cloudera.com/runtime/7.3.1/public-release-notes/topics/rt-runtime-component-versions731100.html>
- [60] H. Fu, M. Sun, B. He, J. Li, and X. Zhu, "A survey of traffic shaping technology in Internet of Things," *IEEE Access*, vol. 11, pp. 3794–3809, 2023, doi: [10.1109/ACCESS.2022.3233394](https://doi.org/10.1109/ACCESS.2022.3233394).
- [61] *Linux Traffic Control: Hierarchy Token Bucket (HTB)*. Accessed: Apr. 12, 2026. [Online]. Available: <https://man7.org/linux/man-pages/man8/tc-htb.8.html>
- [62] *Linux Traffic Control*. Accessed: Apr. 12, 2026. [Online]. Available: <https://man7.org/linux/man-pages/man8/tc.8.html>
- [63] Amazon Web Services. *Amazon EC2 Instance Types-General Purpose Instances*. Accessed: Apr. 12, 2026. [Online]. Available: <https://docs.aws.amazon.com/ec2/latest/instancetype/>
- [64] A. Jurgelionis, J. Laulajainen, M. Hirvonen, and A. I. Wang, "An empirical study of NetEm network emulation functionalities," in *Proc. 20th Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2011, pp. 1–6, doi: [10.1109/ICCCN.2011.6005933](https://doi.org/10.1109/ICCCN.2011.6005933).
- [65] Amazon Web Services. *AWS Direct Connect*. Accessed: Apr. 12, 2026. [Online]. Available: <https://aws.amazon.com/directconnect/>
- [66] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *Proc. IEEE 26th Int. Conf. Data Eng. Workshops (ICDEW)*, Mar. 2010, pp. 41–51, doi: [10.1109/ICDEW.2010.5452747](https://doi.org/10.1109/ICDEW.2010.5452747).
- [67] R. Nambiar and M. Poess, "The making of TPC-DS," in *Proc. 32nd Int. Conf. Very Large Data Bases (VLDB)*, Sep. 2010, pp. 1049–1058.
- [68] *TPC-DS Query 28*. Accessed: Apr. 12, 2026. [Online]. Available: <https://github.com/cloudera/impala-tpcds-kit/blob/master/query-templates/query28.tpl>
- [69] *TPC-DS Query 85*. Accessed: Apr. 12, 2026. [Online]. Available: <https://github.com/cloudera/impala-tpcds-kit/blob/master/query-templates/query85.tpl>
- [70] Cloudera. *TPC-DS Query 67*. Accessed: Apr. 12, 2026. [Online]. Available: <https://github.com/cloudera/impala-tpcds-kit/blob/master/query-templates/query67.tpl>
- [71] R. M. O'Brien, "A caution regarding rules of thumb for variance inflation factors," *Qual. & Quantity*, vol. 41, no. 5, pp. 673–690, Oct. 2007, doi: [10.1007/s11135-006-9018-6](https://doi.org/10.1007/s11135-006-9018-6).
- [72] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola, *Global Sensitivity Analysis: The Primer*. Hoboken, NJ, USA: Wiley, 2008.
- [73] Amazon Web Services. *Direct Connect Virtual Interfaces*. Accessed: Apr. 12, 2026. [Online]. Available: <https://docs.aws.amazon.com/enus/directconnect/latest/UserGuide/create-vif.html>
- [74] L. Guo and J. Y. B. Lee, "Stateful-TCP—A new approach to accelerate TCP slow-start," *IEEE Access*, vol. 8, pp. 195955–195970, 2020, doi: [10.1109/ACCESS.2020.3034129](https://doi.org/10.1109/ACCESS.2020.3034129).
- [75] J. R  th and O. Hohlfeld, "Demystifying TCP initial window configurations of content distribution networks," in *Proc. Netw. Traffic Meas. Anal. Conf. (TMA)*, Jun. 2018, pp. 1–8, doi: [10.23919/TMA.2018.8506549](https://doi.org/10.23919/TMA.2018.8506549).



ATTILA KANTO received the M.Sc. degree in electrical engineering from Budapest University of Technology and Economics. He is currently pursuing the Ph.D. degree in applied informatics with the Doctoral School of Applied Informatics and Applied Mathematics (AIAMDI),   buda University, Hungary. His doctoral research focuses on performance modeling and optimization of large-scale data analytics workloads in hybrid cloud environments. He is also a Senior Director of Engineering at Cloudera, where he leads the design and implementation of distributed data platforms and works on architectural design and system-level performance optimization. He also serves as a Lead Researcher of a nationally funded research project supported by Hungarian National Research, Development and Innovation Office, focusing on workload portability and elasticity across on-premises and public cloud infrastructures. His research interests include hybrid cloud systems, distributed computing, big data performance modeling, and workload execution-time prediction.



ATTILA CSABA MAROSI received the M.Sc. and Ph.D. degrees from Budapest University of Technology and Economics. He is currently a Senior Research Fellow with the Laboratory of Parallel and Distributed Systems (LPDS), Institute for Computer Science and Control (SZTAKI), Hungarian Research Network (HUN-REN), and a Lecturer with the John von Neumann Faculty of Informatics,   buda University, Hungary. With more than 25 years of research and development experience in both industry and academia, he has worked on a broad range of distributed and parallel systems. He has contributed to numerous national and international research and development projects and he has co-authored more than 50 scientific papers. His research interests include the collection and inference of large-scale data.

...