Eurasia Journal of Science and Technology
Print ISSN: 2663-1024
Online ISSN: 2663-1016
DOI: https://doi.org/10.61784/ejst4161

AERIS: ADAPTIVE EXECUTOR-SCOPED RESOURCE MANAGEMENT FOR PRIVACY-PRESERVING DATA PROCESSING IN APACHE SPARK

YiNuo Wang, ChangHao Zhang*
The University of Chicago, 5801 South Ellis Avenue, Chicago, IL 60637, United States.
**Corresponding Author: ChangHao Zhang*

Abstract: Privacy-preserving Spark pipelines introduce executor-local costs that conventional resource policies do not model: cryptographic context reconstruction, enlarged protected working sets, and encrypted spill. This paper presents AERIS, an adaptive executor-scoped controller that couples task memory grants with soft affinity to reusable privacy state. AERIS operates only on coarse metadata—partition size, operator class, key domain, and executor load—and therefore does not inspect plaintext, keys, or differentially private outputs. Its cost estimator is calibrated by real local microbenchmarks of AES-GCM, HMAC-SHA256, bounded Laplace-noised aggregation, scrypt context setup, and encrypted temporary-file spill. We evaluate the design with an event-driven prototype aligned with Spark stage, task, barrier, and executor semantics. The prototype uses the real Wisconsin Diagnostic Breast Cancer dataset and deterministic bootstrap expansion to generate balanced, skewed, and bursty traces. Across ten paired seeds, AERIS reduces makespan by 9.3%, 16.6%, and 10.4% relative to fixed uniform executor budgets. In the two memory-stressed workloads it reduces encrypted spill volume by 62.7% and 64.4%. Against a stronger reactive load-aware baseline, AERIS is 1.5% faster on balanced traces, 0.6% slower on skewed traces, and 5.5% slower on bursty traces, while reducing encrypted spill by 74.4% and 81.0% and lowering p95 service time by 1.0%–3.7%. Queue-inclusive p95 completion latency worsens under skew because longest-first dispatch and state reuse delay short tasks; this limitation is reported rather than hidden. The results support executor-scoped privacy state as a useful scheduling dimension, while also showing that spill avoidance and completion-tail objectives must be balanced explicitly. The artifact is a measurement-backed, trace-driven prototype, not a claim of measurements from a patched Spark cluster.
Keywords: Apache spark; Privacy-preserving analytics; Resource management; Executor affinity; Encrypted spill; Differential privacy

1 INTRODUCTION

Apache Spark established an influential execution model for iterative and interactive analytics by combining fault-tolerant distributed datasets with stage-oriented scheduling [1-2]. Spark SQL and Structured Streaming subsequently extended this model to relational and incremental workloads [3-4]. Resource management in these systems is normally expressed through executor counts, cores, heap or off-heap memory, and task placement. Those dimensions are necessary, but they are incomplete when a pipeline protects data while it is being transformed.

A privacy-preserving task can carry at least three executor-local costs. First, a task may require a key schedule, pseudonymization secret, trusted context, or privacy-accounting state that can be reused only if related work remains on the same executor. Second, encryption, authenticated buffers, clipping, contribution bounding, and intermediate representations increase the protected working set. Third, memory pressure may create encrypted spill, whose encryption, storage, and verification cost is qualitatively different from ordinary in-memory execution. Existing cluster schedulers have studied fairness, short-job latency, learning-based job ordering, and configuration tuning [5-11], while privacy systems have studied encrypted query processing, trusted execution, obliviousness, and differentially private query semantics [12-25]. The executor-scoped coupling between reusable privacy state and resource allocation remains comparatively underexplored.

This paper asks a narrow systems question: can a scheduler reduce privacy-specific overhead by treating executor state and memory as one decision, without changing the privacy mechanism? We answer with AERIS, an adaptive executor-scoped resource and isolation scheduler. AERIS allocates a per-task memory grant from partition size and operator intensity, then chooses among idle executors using predicted service cost, a soft credit for an already available key domain, a context-switch penalty, and a cumulative-load regularizer. The controller uses metadata only. It does not adjust epsilon, clipping bounds, encryption algorithms, or the data values being processed.

The evaluation is deliberately reproducible and bounded in its claims. The available execution environment did not contain an Apache Spark runtime or a multi-node cluster, so we do not label the results as cluster deployment measurements. Instead, we measure real privacy kernels and secure spill locally, fit a transparent linear cost model, and execute deterministic Spark-style traces in an event-driven simulator that enforces stages, executor occupancy, memory reservation, and barriers. This methodology supports controlled paired comparisons, but cannot reproduce network shuffle, JVM garbage collection, speculative execution, or failure recovery.

The contributions are:

1) an executor-scoped resource model that exposes privacy-state reuse, protected working-set demand, and encrypted spill as scheduling variables;
2) a lightweight adaptive policy combining memory grants, soft key affinity, operation-switch cost, and load regularization without exposing sensitive data;
3) a measurement-backed trace methodology using real AES-GCM, HMAC-SHA256, differential-privacy, key-setup, and encrypted-spill kernels; and
4) an honest empirical characterization of the benefit–cost frontier: strong gains over uniform allocation, major spill reductions against a reactive baseline, and a measurable queue-tail penalty under skew.

2 BACKGROUND AND MOTIVATION

2.1 Spark Execution and Resource Decisions

Spark decomposes a job into stages separated by shuffle dependencies; each stage contains tasks that operate on partitions. Executors host task slots and maintain process-local state, while the driver coordinates scheduling. RDD lineage supplies fault recovery [2], Spark SQL adds a declarative optimizer and compact execution engine [3], and Structured Streaming maps incremental queries into a declarative model [4]. Drizzle further shows that scheduling granularity and coordination paths materially affect latency [5]. A practical controller can therefore act at task dispatch without redesigning the entire programming model.
Classic resource-management work separates several concerns. Mesos exposes fine-grained cluster resources [6]. Dominant Resource Fairness provides a principled multi-resource allocation rule [7]. Sparrow targets distributed low-latency scheduling [8], Decima learns job-level scheduling decisions [9], Ernest predicts analytics performance from sampled runs [10], and Rover tunes Spark SQL configurations using online transfer [11]. These systems motivate adaptive and measurement-driven control, yet their task descriptors do not directly represent reusable cryptographic state or the cost of protected spill.

2.2 Privacy-Preserving Analytics

System-level confidentiality has been pursued through trusted execution and encrypted processing. Opaque combines encrypted distributed analytics with oblivious execution [12]. VC3, SCONE, and Ryoan use trusted hardware or constrained execution to protect cloud computation [13-15]. Seabed, CryptDB, and Monomi process encrypted data using specialized representations and query plans [16-18]. These techniques differ in leakage model and trusted computing base, but all can introduce state setup, enlarged intermediates, and nonuniform operator costs.
Differential privacy provides a semantic guarantee about the sensitivity of released results [19]. PINQ, Airavat, GUPT, bounded-contribution SQL, Chorus, and PrivateSQL progressively integrate differential privacy with scalable data processing [20-25]. A scheduler must not silently alter their mechanism parameters: changing epsilon, contribution bounds, clipping, or noise calibration would change the guarantee. AERIS therefore treats privacy semantics as invariants and optimizes only resource placement and execution overhead.

2.3 Motivating Microbenchmarks

The measured kernels show why a single “rows processed” estimate is insufficient. At 32,000 rows, AES-GCM takes 6.34 ms, HMAC pseudonymization takes 126.46 ms, and the combined kernel takes 133.73 ms on the test host. The median scrypt setup cost is 8.22 ms. Encrypted spill is also size dependent: the median for a 16 MB payload ranges from 34.71 to 51.59 ms across segmentations. These are real local timings, not values selected to make the scheduler favorable (Figure 1).

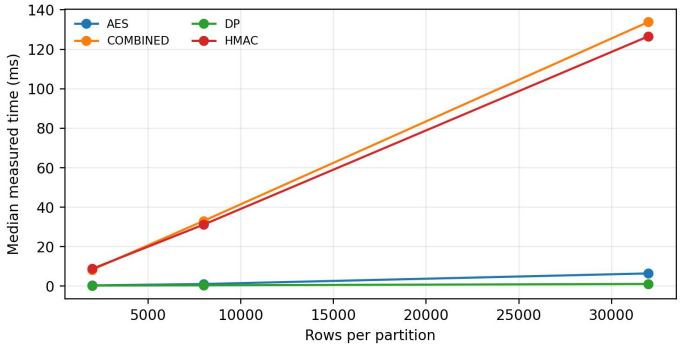

Figure 1 Measured Privacy-Kernel Scaling
Note: Points are medians of seven repetitions; logarithmic scaling is not used.

3 PROBLEM FORMULATION AND THREAT MODEL

3.1 Task and Executor State

For each ready task t, the controller observes a metadata descriptor x_t = (n_t, o_t, d_t, q_t): partition row count n_t, operator class o_t, key domain d_t, and stage identifier q_t. The operator class is one of DP aggregation, HMAC pseudonymization, AES-GCM protection, or a combined transform in the prototype. The key domain is an opaque identifier; it is not a cryptographic key. Each executor e exposes s_e = (a_e, C_e, r_e, l_e): next available time, a bounded least-recently-used set of key-domain identifiers, an exponentially smoothed observed milliseconds-per-MB rate, and cumulative busy time.

A memory grant g_t determines whether the protected working set fits. For a task data size D_t and operator intensity c(o_t), the prototype estimates the working set as D_t(0.90 + 0.35c). Overflow beyond g_t is counted as encrypted spill. The global constraint is that concurrent grants never exceed M = 72 MB and every admitted task receives at least 6 MB. This creates a genuine allocation problem: a large grant can suppress spill but may reduce concurrency.

$$\text{sum of active grants } g_t \le M, \text{ and } 6\ \text{MB} \le g_t \le 1.8(M/|E|) \quad (1)$$

3.2 Design Objective

The design target is multi-objective rather than a claim of universally minimum latency. It seeks low stage makespan, low encrypted-spill volume, few privacy-context reconstructions, and balanced executor use. These terms can conflict. In particular, longest-first dispatch shortens the tail of a stage but can increase the completion time of short tasks waiting behind long tasks. We therefore report both task service time and queue-inclusive completion latency, and we avoid collapsing all outcomes into a single opaque score.

3.3 Threat Model and Privacy Invariants

The threat model is intentionally limited. We assume the Spark driver and executor software that implements the selected privacy mechanism is trusted to follow its specification. Storage used for spill is considered untrusted, so spill payloads are encrypted and authenticated with AES-GCM. Network attackers, malicious operating systems, side channels, denial of service, and compromised trusted-execution hardware are outside this prototype. The controller itself receives only coarse metadata and aggregate state. It never receives plaintext records, raw features, cryptographic key material, HMAC outputs, DP-noised answers, or per-record contribution information.

AERIS preserves four invariants: the privacy mechanism and parameters are unchanged; ciphertexts and spill files use the same authenticated-encryption primitive irrespective of scheduling; key-domain identifiers are opaque; and task reordering occurs only within a stage whose tasks are otherwise independent. Consequently, the controller changes performance behavior, not the mathematical definition of the DP query or the cryptographic algorithm. This is a systems noninterference argument at the control interface, not a proof against side channels (Figure 2).

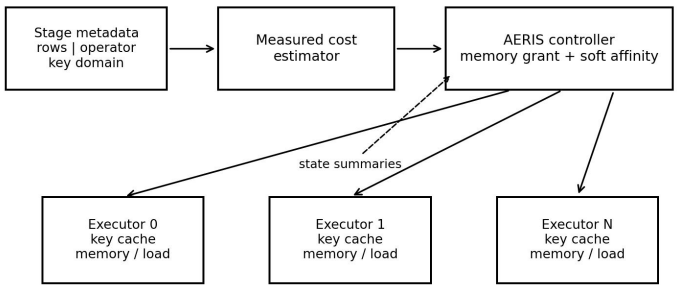

Figure 2 AERIS Control Path

Note: Sensitive values stay inside executor-local privacy operators.

4 AERIS DESIGN

4.1 Measurement-Backed Cost Estimator

For each operator, AERIS fits a nonnegative affine model b_o(n) = alpha_o + beta_o n to the median measured microbenchmark times. Run-to-run variation is represented with a log-normal factor whose sigma is derived from the measured coefficient of variation. A separate affine model estimates encrypted-spill time from overflow MB and segment count. The estimator is deliberately simple: three row sizes provide too little evidence for a complex learner, while an interpretable model makes every simulated cost traceable to a measured primitive.

$$\text{predicted\_service} = (b_o(n) + \text{key\_miss\_cost} + \text{spill\_cost}) * \text{measured\_noise} \quad (2)$$

The current calibration gives a median key-setup cost of 8.215 ms and a fitted spill slope of 2.736 ms/MB. The segment coefficient is clipped to zero because the least-squares fit is nonpositive after controlling for payload size. This is not interpreted as proof that segmentation is free; it only prevents the small local sample from introducing a negative cost.

4.2 Adaptive Memory Grant

AERIS computes a grant from the task data size and operator intensity. In the prototype, c is 0.25 for DP, 0.70 for HMAC, 1.00 for AES, and 1.35 for the combined transform. The resulting rule is a bounded engineering heuristic, not a trained model. Its purpose is to give protected, high-intensity tasks more memory while leaving room for concurrent tasks. The fixed-memory ablation replaces this grant with 12 MB for every task.

$$g_t = \mathrm{clip}(5 + 0.55\, D_t(0.65 + 0.35c) + 2.5c,\ 6,\ 1.8M/|E|) \tag{3}$$

4.3 Soft Privacy-State Affinity

A key domain represents a reusable executor-local context. A strict affinity rule can overload a favored executor, so AERIS treats locality as a credit rather than a constraint. For the longest predicted ready task, it evaluates every idle executor. The score includes predicted service with the candidate grant, a 3% operation-switch penalty when the previous operator differs, 0.12 times cumulative busy time, and a credit equal to the measured key-setup cost when the domain is already cached. The minimum-score feasible executor is selected.

$$\mathrm{score}(t,e) = p(t,e,g) + \mathrm{switch}(t,e) + 0.12\, l_e - \mathrm{cached}(d_t,e)*k \tag{4}$$

Algorithm 1 AERIS Dispatch Decision for One Idle-Executor Event

```
Input: ready tasks T, idle executors E, memory M
sort T by predicted base time (descending)
t <- first task in T
for e in E:
    g <- adaptive_grant(t,e,M)
    miss <- domain(t) not in cache(e)
    score <- predicted_service(t,g,miss)
                + switch_penalty + 0.12*busy(e)
                - cached_key_credit
choose feasible e with minimum score
reserve g; update cache and feedback on finish
```

4.4 Complexity and Integration Point

Sorting a stage with N tasks costs O(N log N). Each dispatch evaluates at most |E| idle executors, so the incremental decision cost is O(|E|) and uses constant-size state per executor apart from the bounded key-domain cache. A Spark implementation could place the policy between TaskScheduler-level placement and an executor-side memory broker, with event-listener feedback updating observed service rate and context identifiers. The prototype does not patch these classes; this integration sketch identifies a plausible path rather than reporting code that was not built.

5 PROTOTYPE AND EXPERIMENTAL METHOD

5.1 Implementation Scope

The artifact is implemented in Python and consists of real microbenchmarks, a deterministic trace generator, and an event-driven scheduler. Each task reserves one executor and a memory grant until completion. Stages have barriers: the next stage is released only after all tasks in the current stage finish. The simulator maintains executor availability, memory reservations, key-domain LRU state, operation history, and measured service-rate feedback. It records every task start, finish, queue time, service time, grant, cache event, and spill amount.

5.2 Dataset and Privacy Operators

The source data are the Wisconsin Diagnostic Breast Cancer dataset from the UCI Machine Learning Repository, loaded through scikit-learn [26-27]. It contains 569 observations, 30 real-valued features, and two diagnostic classes. For a requested partition size, rows are sampled with replacement under a fixed seed and multiplied by independent Gaussian factors with mean 1.0 and standard deviation 0.002. This expansion is used only to scale kernel work; it is not presented as a new clinical dataset and no scientific inference is drawn from the perturbed records.

The AES kernel serializes the 30 float32 features, encrypts and decrypts the partition with AES-GCM, and verifies the round trip under a fixed associated-data label, consistent with the authenticated-encryption construction specified by FIPS 197 and NIST SP 800-38D [28-29]. The HMAC kernel creates a 16-byte truncated HMAC-SHA256 pseudonym for each row identifier. The DP kernel clips the first feature to 0–40, computes a mean for each class, and adds Laplace noise with scale 40/count, corresponding to epsilon = 1 for the bounded mean implementation. The combined operator executes HMAC, AES-GCM, and DP aggregation in sequence.

5.3 Workloads, Baselines, and Configuration

Table 1 Trace Workloads (24 Tasks per Stage)

Trace	Tasks	Stages and operators	sigma; mean rows/stage
Balanced	72	3: HMAC -> AES -> Combined	0.25; 18k / 24k / 20k
Skewed	72	3: HMAC -> AES -> Combined	0.75; 18k / 30k / 24k
Bursty	96	4: DP -> Combined -> AES -> Combined	0.55; 12k / 34k / 16k / 38k

Partition sizes are log-normally distributed and clipped to 2,000–150,000 rows (Table 1). Four key domains are sampled with probabilities 0.46, 0.26, 0.18, and 0.10. All policies use six executors, 72 MB total logical memory, a 6 MB minimum grant, and a two-domain executor cache. Ten independent seeds produce paired traces for each workload.
Uniform is a fixed 12 MB, FIFO, round-robin policy. Reactive is a strong load-aware baseline: it uses longest-first task order, places work on the idle executor with the lowest observed ms/MB, and changes the grant by ±25%–35% when an executor differs sufficiently from the median rate. AERIS uses the policy in Algorithm 1. AERIS-Fixed retains state-aware placement but fixes memory at 12 MB. AERIS-NoAffinity removes the key-cache credit.

5.4 Metrics and Statistical Protocol

Makespan is the finish time of the final task. Throughput is total rows divided by makespan. Completion latency is finish time minus stage release and therefore includes queueing; service time excludes queueing. We report p95 for both. Privacy overhead metrics are key-cache misses, encrypted overflow segments, and spilled MB. Executor fairness uses Jain's index over cumulative service time. Aggregate tables report the mean and an uncorrected 95% confidence interval, 1.96s/sqrt(10). Paired percentage changes are computed seed by seed. A one-sided paired Wilcoxon signed-rank test evaluates whether the baseline metric is greater than AERIS. With only ten seeds, p-values are descriptive and should be interpreted alongside effect sizes.

5.5 Execution Environment

The recorded environment is Python 3.13.5 on Linux-6.12.13-x86_64-with-glibc2.41, with 5 visible CPUs, NumPy 2.3.5, pandas 2.2.3, scikit-learn 1.8.0, and cryptography 46.0.4. Kernel measurements use seven repetitions per row size and operator; scrypt uses eleven; encrypted spill uses five repetitions per payload/segmentation pair. The environment manifest and raw CSV files are included in the artifact.

6 RESULTS

6.1 Measured Primitive Costs

Table 2 Local Privacy-Primitive Measurements

Primitive (32k rows)	Median ms	P95 ms	MB/s
DP	1.01	1.29	3643.1
HMAC	126.46	140.97	29.0
AES	6.34	6.62	577.6
COMBINED	133.73	142.59	27.4
scrypt setup	8.22	8.73	--

HMAC dominates the combined task at large partitions (Table 2): the 32k-row HMAC median is 126.46 ms and the combined median is 133.73 ms, whereas AES-GCM is 6.34 ms. The relative cost is specific to this implementation and hardware; AERIS does not assume it holds universally. The fitted estimator therefore remains replaceable with measurements from the deployment platform.

6.2 End-to-End Makespan

Against uniform allocation, AERIS reduces paired makespan by 9.34% in balanced traces, 16.62% in skewed traces, and 10.41% in bursty traces. The one-sided Wilcoxon p-value is 0.000977 in all three comparisons. The improvement comes from longest-first dispatch, lower context reconstruction, and targeted memory grants. This establishes the primary claim: executor-scoped adaptation is materially better than a fixed round-robin policy in the evaluated model.
The reactive baseline is much harder to beat because it already uses longest-first dispatch and feedback. AERIS is 1.54% faster on balanced traces ($p = 0.000977$), 0.59% slower on skewed traces, and 5.48% slower on bursty traces. The latter two differences show that memory grants optimized for spill reduction can reduce concurrency and increase makespan. AERIS should therefore not be described as universally faster than a strong load-aware scheduler (Table 3; Figure 3-4).

Table 3 Main Results over Ten Paired Seeds

Trace	Policy	Makespan ms (95% CI)	P95 service	P95 completion	Spill MB
Bal.	Uniform	787.4 +/- 34.3	112.6	348.2	0.00
Bal.	Reactive	723.2 +/- 15.5	113.7	357.6	0.00
Bal.	AERIS	712.1 +/- 15.5	112.2	355.1	0.00
Skew.	Uniform	1207.6 +/- 148.7	225.5	499.9	16.12
Skew.	Reactive	988.2 +/- 70.8	227.0	543.1	21.90
Skew.	AERIS	995.5 +/- 84.4	224.6	566.6	7.92
Burst.	Uniform	1814.8 +/- 90.4	296.9	710.3	19.30
Burst.	Reactive	1538.3 +/- 83.2	307.4	775.4	36.26
Burst.	AERIS	1624.1 +/- 100.0	295.7	837.9	7.31

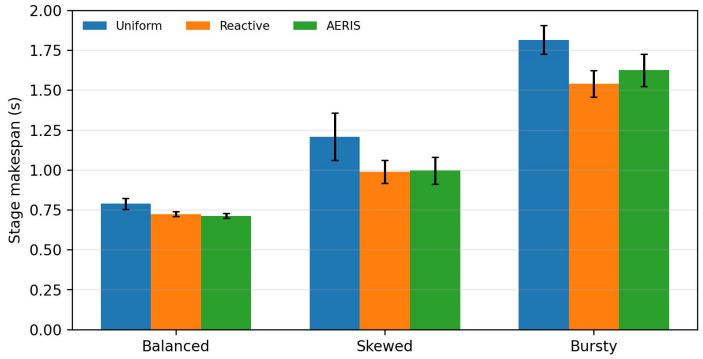

Figure 3 Mean Stage Makespan with 95% Confidence Intervals

6.3 Encrypted Spill and Privacy-State Reuse

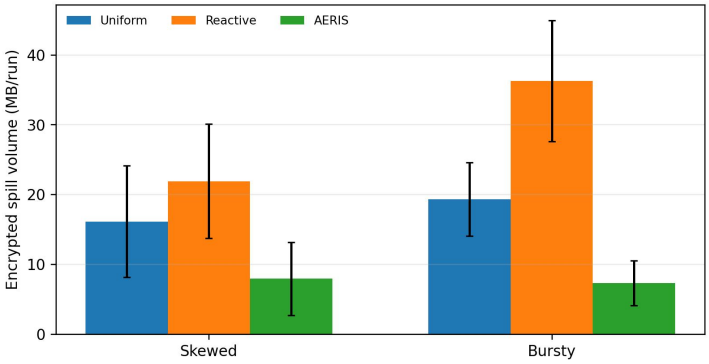

Figure 4 Encrypted Spill Volume in Memory-Stressed Traces
Note: Error bars are 95% confidence intervals.

Spill is the clearest privacy-specific benefit. Relative to uniform allocation, AERIS reduces spilled MB by 62.65% on skewed traces and 64.42% on bursty traces. Relative to reactive allocation, reductions are 74.40% and 80.97%. All four one-sided Wilcoxon tests have p <= 0.001953. AERIS also reduces overflow segment counts by 53.10% and 68.49% against reactive. Because the local spill benchmark performs authenticated encryption, file I/O, re-reading, and decryption, the reduction represents avoided measured work rather than an arbitrary simulator penalty.

Key-cache behavior is more nuanced. AERIS reduces misses by 16.92% in balanced traces against reactive (p = 0.000977), by 1.01% in skewed traces (not significant), and by 9.12% in bursty traces (p = 0.0283). The no-affinity ablation is nearly identical to AERIS in makespan and spill. This indicates that the 8.2 ms setup penalty is too small relative to HMAC-heavy service time for key affinity to dominate; the main evaluated value of AERIS comes from adaptive memory and load-aware placement, not from an exaggerated locality claim.

6.4 Tail Behavior and Tradeoff

AERIS lowers p95 service time against reactive by 1.40% in balanced traces, 1.03% in skewed traces, and 3.66% in bursty traces. However, queue-inclusive p95 completion latency moves in the opposite direction under skew: AERIS is 3.92% worse than reactive on skewed traces and 7.93% worse on bursty traces. Relative to uniform, it is 12.59% and 17.98% worse. Longest-first dispatch causes short tasks to wait, and soft affinity/load scoring can preserve a slightly slower placement to avoid later privacy overhead (Figure 5).

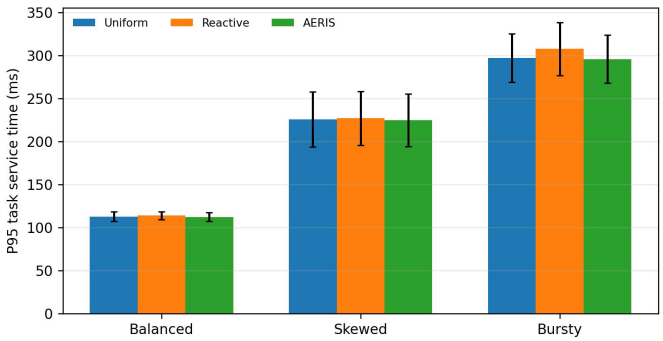

Figure 5 P95 Task Service Time, Excluding Scheduler Queueing

This divergence is important for system design. Service tail measures the efficiency of running tasks after admission; completion tail measures user-visible waiting. A production controller should expose a policy knob or constrained optimizer that limits completion-tail regression while assigning a value to avoided encrypted spill. The present controller demonstrates the resource dimension but does not solve that multi-objective tuning problem.

6.5 Ablation and Resource Safety

Table 4 Ablation: Positive Values Favor Full AERIS

Trace	Comparison	Makespan gain	Spill reduction	Miss reduction
Skewed	AERIS vs fixed grant	-1.17%	+62.65%	-2.60%
Bursty	AERIS vs fixed grant	-6.42%	+64.42%	+1.18%
Skewed	AERIS vs no affinity	0.00%	0.00%	0.00%
Bursty	AERIS vs no affinity	+0.00%	0.00%	+0.51%

The adaptive grant reduces spill by more than 62% but is slower than the fixed-grant AERIS variant by 1.17% on skewed traces and 6.42% on bursty traces (Table 4). This confirms that the grant trades concurrency for spill avoidance. The controller respects the 72 MB global bound in every run; mean peak allocation is 71.87 MB for skewed AERIS and 72 MB for bursty AERIS. Jain fairness remains above 0.995 on average, so the state-aware score does not create material executor starvation in these traces.

7 DISCUSSION

7.1 What the Results Establish

The evidence supports three bounded conclusions. First, uniform executor budgets leave measurable performance on the table when privacy operators have heterogeneous working sets and setup costs. Second, encrypted spill can be reduced substantially by task-specific grants even when raw makespan does not improve against a strong scheduler. Third, privacy-state locality is not automatically the dominant lever: its importance depends on the ratio between context setup and operator service time. A credible scheduler should measure that ratio rather than presume it.

These conclusions motivate treating secure spill volume as a first-class operational metric. Ordinary schedulers typically regard spill as a performance symptom. In a protected pipeline, each spilled byte also traverses an encryption and integrity boundary, creates additional ciphertext lifetime, and may interact with storage-key management. Reducing spill is therefore valuable even when makespan is near parity, provided the memory and completion-tail costs are visible.

7.2 Spark Integration Considerations

A practical Spark implementation would require careful ownership of memory and state. The driver could attach an opaque privacy profile to TaskSet metadata. Executors could report coarse context identifiers, protected-working-set pressure, and smoothed service rate through heartbeat extensions. A memory broker would reserve off-heap or executor-managed capacity before task launch and release it on completion. Spill encryption should remain inside the executor operator, not the scheduler. Failure recovery must invalidate or reconstruct cached contexts, and speculative copies must not double-spend a DP privacy budget. These requirements are engineering work left to a real Spark patch.

7.3 Policy Improvement

The observed queue-tail regression suggests a direct next step: replace pure longest-first order with a constrained priority such as predicted remaining stage tail plus aging, and reject a memory grant when its predicted spill saving does not compensate for the concurrency loss. A controller could learn the tradeoff online, but any learned policy should retain explicit safety constraints on total memory, maximum short-task wait, and privacy-parameter invariance. The current heuristic is useful precisely because its failure mode is interpretable.

8 LIMITATIONS AND THREATS TO VALIDITY

The principal limitation is external validity. This is a trace-driven prototype calibrated by local measurements, not a deployment on Apache Spark. It omits JVM allocation, garbage collection, code generation, shuffle networking, disk contention among executors, speculative execution, executor loss, dynamic allocation, and heterogeneous machines. The 72 MB memory pool is a logical stress parameter chosen to induce controlled overflow; it is not meant to imitate the heap size of a production executor.

The data source is real but small and tabular. Bootstrap expansion preserves the computational shape of 30 float features, not the distributional complexity of production logs, images, or joins. Only four operator profiles and four key domains are modeled. The DP operation is a bounded two-group mean at epsilon = 1; the experiment does not measure privacy utility or composition across repeated analyst queries. HMAC truncation and deterministic test keys are benchmark choices, not a deployment key-management design.

The cost model uses only three row sizes and a simple affine fit. Secure spill timings are variable because temporary file behavior depends on the host filesystem and cache. The simulator multiplies service cost by measured log-normal noise but does not model correlated interference. Ten seeds permit paired tests but do not support fine tail-probability estimation. Confidence intervals are uncorrected for multiple comparisons. Finally, the controller uses task row count and operator type, which may themselves be sensitive metadata in some threat models; deployment would need bucketing, access control, or privacy-preserving telemetry.

These limitations do not invalidate the measured primitive timings or the deterministic policy comparison, but they bound the paper's claim: AERIS is evidence that executor-scoped privacy state is a promising resource dimension and a reproducible design hypothesis for a Spark implementation, not proof of production-scale superiority.

9 RELATED WORK

AERIS draws from three lines of work. Distributed dataflow and cluster scheduling establish stage-oriented execution, fine-grained resource sharing, fairness, randomized low-latency placement, learned scheduling, and performance prediction [1-10]. Rover is especially relevant because it shows that Spark configuration can be adapted online from prior knowledge and observed workloads [11]. AERIS differs by making privacy-state reuse and encrypted spill explicit at executor scope rather than tuning a job-level configuration vector.

Confidential analytics systems protect computation using enclaves, oblivious operators, or encrypted query processing [12-18]. Their mechanisms often require specialized state, memory, and I/O, but their primary contribution is the security architecture rather than adaptive Spark executor allocation. AERIS is complementary: it does not replace the confidentiality mechanism and could, after implementation-specific validation, schedule tasks that invoke such mechanisms.

Differentially private systems integrate formal release guarantees into dataflow, MapReduce, SQL, and scalable mechanism frameworks [19-25]. Their correctness depends on contribution bounds, sensitivity, composition, and noise. AERIS explicitly refuses to tune those semantics. Its role is to reduce execution overhead while preserving the selected mechanism. This separation distinguishes resource adaptation from privacy-budget optimization.

10 CONCLUSION

This paper introduced AERIS, an executor-scoped controller for privacy-preserving Spark-style analytics. AERIS combines measured cost estimation, adaptive memory grants, soft privacy-state affinity, operation-switch awareness, and load regularization. In a reproducible trace-driven evaluation calibrated by real privacy kernels, it reduces makespan by 9.3%–16.6% against uniform allocation and reduces encrypted spill by 62.7%–64.4% in memory-stressed traces. Against a stronger reactive scheduler it preserves near-parity to 5.5% worse makespan while avoiding 74.4%–81.0% of encrypted spill. The same experiments reveal a queue-inclusive p95 completion penalty under skew and a modest role for key affinity on the measured host. The central lesson is therefore not that one heuristic dominates every scheduler, but that executor-local privacy state and protected spill should be visible resource-management signals. A full Spark implementation should retain these signals while adding a hard completion-tail constraint and cluster-level validation.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

REFERENCES

[1] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster Computing with Working Sets. Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud), 2010.

[2] Zaharia M. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2012: 15-28.

[3] Armbrust M. Spark SQL: Relational Data Processing in Spark. Proceedings of the ACM SIGMOD International Conference on Management of Data, 2015: 1383-1394. DOI: 10.1145/2723372.2742797.
[4] Armbrust M. Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark. Proceedings of the ACM SIGMOD International Conference on Management of Data, 2018: 601-613. DOI: 10.1145/3183713.3190664.
[5] Venkataraman S, Panda A, Ousterhout K, et al. Drizzle: Fast and Adaptable Stream Processing at Scale. Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP), 2017: 374-389. DOI: 10.1145/3132747.3132750.
[6] Hindman B. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2011: 295-308.
[7] Ghodsi A, Zaharia M, Hindman B, et al. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2011: 323-336.
[8] Ousterhout K, Wendell P, Zaharia M, et al. Sparrow: Distributed, Low Latency Scheduling. Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP), 2013: 69-84. DOI: 10.1145/2517349.2522716.
[9] Mao H, Schwarzkopf M, Venkatakrishnan S B, et al. Learning Scheduling Algorithms for Data Processing Clusters. Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM), 2019: 270-288. DOI: 10.1145/3341302.3342080.
[10] Venkataraman S, Yang Z, Franklin M, et al. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2016: 363-378.
[11] Shen Y, Ren X, Lu Y, et al. Rover: An Online Spark SQL Tuning Service via Generalized Transfer Learning. Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2023: 4800-4812. DOI: 10.1145/3580305.3599953.
[12] Zheng W. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2017: 283-298.
[13] Schuster F. VC3: Trustworthy Data Analytics in the Cloud Using SGX. Proceedings of the IEEE Symposium on Security and Privacy, 2015: 38-54. DOI: 10.1109/SP.2015.10.
[14] Arnautov S. SCONE: Secure Linux Containers with Intel SGX. Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2016: 689-703.
[15] Hunt T. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data. Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2016: 533-549.
[16] Papadimitriou A. Big Data Analytics over Encrypted Datasets with Seabed. Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2016: 587-602.
[17] Popa R A, Redfield C M S, Zeldovich N, et al. CryptDB: Protecting Confidentiality with Encrypted Query Processing. Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP), 2011: 85-100. DOI: 10.1145/2043556.2043566.
[18] Tu S, Kaashoek M F, Madden S, et al. Processing Analytical Queries over Encrypted Data. Proceedings of the VLDB Endowment, 2013, 6(5): 289-300. DOI: 10.14778/2535573.2488336.
[19] Dwork C. Differential Privacy. In: Automata, Languages and Programming, LNCS 4052. Springer, 2006: 1-12. DOI: 10.1007/11787006_1.
[20] McSherry F. Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis. Proceedings of the ACM SIGMOD International Conference on Management of Data, 2009: 19-30. DOI: 10.1145/1559845.1559850.
[21] Roy I, Setty S T V, Kilzer A, et al. Airavat: Security and Privacy for MapReduce. Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2010: 297-312.
[22] Mohan P, Thakurta A, Shi E, et al. GUPT: Privacy Preserving Data Analysis Made Easy. Proceedings of the ACM SIGMOD International Conference on Management of Data, 2012: 349-360. DOI: 10.1145/2213836.2213876.
[23] Wilson R J, Zhang C Y, Lam W, et al. Differentially Private SQL with Bounded User Contribution. Proceedings on Privacy Enhancing Technologies, 2020, 2020(2): 230-250. DOI: 10.2478/popets-2020-0025.
[24] Johnson N M, Near J P, Hellerstein J M, et al. Chorus: A Programming Framework for Building Scalable Differential Privacy Mechanisms. Proceedings of the IEEE European Symposium on Security and Privacy, 2020: 535-551. DOI: 10.1109/EuroSP48549.2020.00041.
[25] Kotsogiannis I, Tao Y, He X, et al. PrivateSQL: A Differentially Private SQL Query Engine. Proceedings of the VLDB Endowment, 2019, 12(11): 1371-1384. DOI: 10.14778/3342263.3342274.
[26] Wolberg W H, Street W N, Mangasarian O L. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1995. DOI: 10.24432/C5DW2B.
[27] Pedregosa F, et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 2011, 12: 2825-2830.
[28] National Institute of Standards and Technology. Advanced Encryption Standard (AES). FIPS PUB 197-upd1, 2023. DOI: 10.6028/NIST.FIPS.197-upd1.
[29] Dworkin M. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D, 2007. DOI: 10.6028/NIST.SP.800-38D.