# An Empirical Study of Inter-cluster Resource Orchestration within Federated Cloud Clusters

Dominic Lindsay
*EDS Lab*
*Lancaster University*
Lancaster, United Kingdom
d.lindsay4@lancaster.ac.uk

Gingfung Yeung
*EDS Lab*
*Lancaster University*
Lancaster, United Kingdom
g.yeung1@lancaster.ac.uk

Yehia Elkhatib
*School of Computing Science*
*Glasgow University*
Glasgow, United Kingdom
yehia.elkhatib@glasgow.ac.uk

Peter Garraghan
*EDS Lab*
*Lancaster University*
Lancaster, United Kingdom
p.garraghan@lancaster.ac.uk

***Abstract*—Federated clusters are composed of multiple independent clusters of machines interconnected by a resource management system, and possess several advantages over centralized cloud datacenter clusters including seamless provisioning of applications across large geographic regions, greater fault tolerance, and increased cluster resource utilization. However, while existing resource management systems for federated clusters are capable of improving application *intra-cluster* performance, they do not capture *inter-cluster* performance in their decision making. This is important given federated clusters must execute a wide variety of applications possessing heterogeneous system architectures, which are a impacted by unique inter-cluster performance conditions such as network latency and localized cluster resource contention. In this work we present an empirical study demonstrating how inter-cluster performance conditions negatively impact federated cluster orchestration systems. We conduct a series of micro-benchmarks under various cluster operational scenarios showing the critical importance in capturing inter-cluster performance for resource orchestration in federated clusters. From this benchmark, we determine precise limitations in existing federated orchestration, and highlight key insights to design future orchestration systems. Findings of notable interest entail different application types exhibiting innate performance affinities across various federated cluster operational conditions, and experience substantial performance degradation from even minor increases to latency (8.7x) and resource contention (12.0x) in comparison to centralized cluster architectures.**

***Index Terms*—Federated cluster computing, Federated orchestration, Cloud federation, Resource management, Scheduling.**

## I. Introduction

It is now commonplace for large technology companies to manage and operate large clusters of machines, that form the backbone of their cloud datacenter infrastructure. To facilitate growing user demand to provision cloud services globally, *federated cluster* environments have formed whereby workloads comprising applications are encapsulated within containers that execute across multiple clusters. These federated clusters are increasingly prominent within industry [5], [27]–[29], and are leveraged to provision cloud applications with higher levels of application performance, service availability, and service reliability for a wide plethora of application types.

Effective workload placement and execution within clusters is made possible via *orchestration*. Encompassing resource management [5], [7], [28] and application scheduling [3], orchestration systems are responsible for abstracting and managing the complexities of workload placement and execution within clusters in a resource-efficient manner. Prominent orchestration systems found within cloud datacenters typically leverage centralized architectures designed to manage clusters connected over non-volatile, high bandwidth substrates [2]. Whilst application framework diversity has continued to increase requiring low node-to-node network latency and high bandwidth, decentralized architectures have been created that execute diverse sets of scheduling policies [5], [29].

*Federated orchestration* systems – whereby individual clusters jointly interact with each other, yet manage and operate their own scheduler and resource pool independently – have been identified as an effective means to facilitate federated workloads [5], [27]. This is due to their ability to manage autonomous clusters – typically characterized by geographically distinct heterogeneous clusters – as a single resource pool. This allows for rapid workload deployment across clusters composed by tens, hundreds, or thousands of machines via offloading placement and resource decision making to localized cluster schedulers resulting in higher performance from reduced head-of-line blocking, scheduling times, and proximity to client devices [3].

Existing orchestration systems for cloud are effective at improving *intra-cluster* (node-to-node) performance. However, they are not innately designed to consider *inter-cluster* (cluster-to-cluster) performance when enacting workload placement and execution decisions. This is problematic as clusters leveraged for cloud computing are exposed to network volatility [3], dynamic utilization [30], and heterogeneous scheduling architectures [7] – all which are intrinsic to federated cluster environments. The majority of federated orchestration systems only consider resource demand and reservation [5], [28], and omit characteristics at network-level (bandwidth, latency), node-level (interference, contention) and cluster-level (scheduler type). Orchestrators that do capture inter-cluster performance are designed for singular application frameworks [8] and are thus not generalizable to the wide range of workloads found within federated cloud environments. Failure to capture and exploit inter-cluster performance in orchestration results in poor application placement decisions, reduced Quality of Service (QoS), and degraded workload performance within federated cluster environments.

This work presents an empirical study demonstrating the challenges of detrimental inter-cluster performance conditions for federated cluster orchestration. We conduct a series of micro-benchmarks of different application types (streaming, machine learning, batch) within a 30-node experimental federated cluster infrastructure under various conditions to determine key factors impacting federated cluster workload performance. Our experiments demonstrate that detrimental federated cluster operational conditions can significantly degrade workload performance, most notably inter-cluster latency (8.7x) and high resource contention (12x). Moreover, we discover that different federated cluster applications exhibit particular tolerances to detrimental federated cluster conditions. From this, we highlight precise research directions and changes in existing resource schedulers required to achieve effective federated cluster orchestration.

## II. Background

### A. Federated Clusters

The previous decade has seen wide spread adoption of cloud datacenters to deliver scalable services across geographic bounds. Such systems are underpinned by networks of clustered machines. Increasingly, the scale of such clusters has grown to such an extent that the use of centralized control plane and resource management for thousands of machines to deliver cloud services globally has become increasingly difficult to attain high quality, high throughput resource placements, degrading application performance. Furthermore, organizational constraints placed on scheduling policies such as resource share allocations lead to pools of unused or underutilized resources limiting cluster utilisation and hardware specialisation.

Federated clusters are networks of decentralized autonomous clusters managed by a distributed control plane. Workloads may be executed across any node within a federation and is perceived as a unified system to an application. Federations offer several advantages over centralized infrastructures including, increased cluster specialization and workload specific customization [27], localized scheduling polices [5], improved cluster utilization, and increased application resilience [19]. It is common for federated clusters operated by companies such as Alibaba, Facebook, and Google to span multiple regions such as those found in Fig. 1 in order to provision cloud services globally.

### B. Orchestration

Orchestration systems are frameworks responsible for controlling cluster compute resources (CPU, disk, memory), and job scheduling [17]. Jobs (comprising tasks encapsulated in containers or VMs) are assigned onto cluster nodes via the *resource scheduler* and *resource manager* which enacts placement and resource allocation decisions for applications. Orchestration are designed with different objectives such as performance, fairness, and utilization.

Orchestration systems can be formed by centralized and decentralized architectures as shown in Fig. 2. A centralized

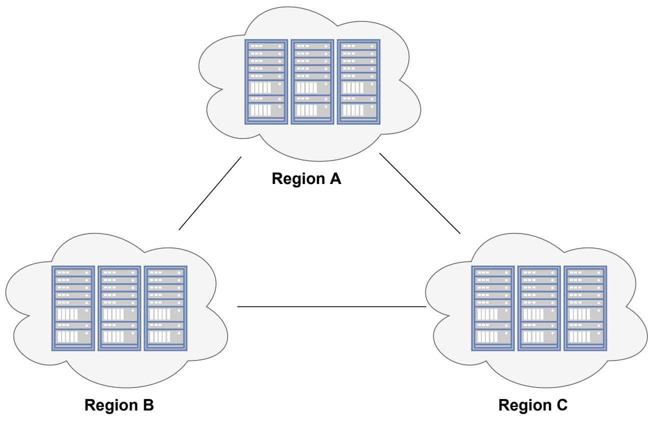


Fig. 1: An overview of a federated cluster environment.

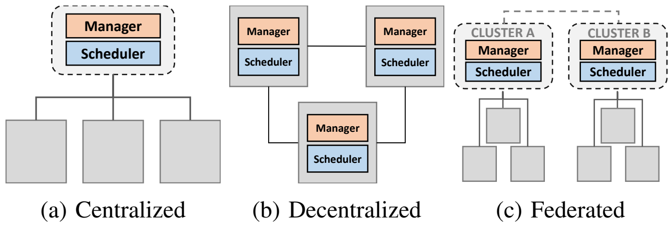


Fig. 2: Comparison of cluster resource orchestration.

model entails the cluster resource manager tracking resource usage, node liveness, and granting application leader resource requests [26]. Decentralized architectures delegate resource management in a hierarchical manner [7]. In contrast, a federated model involves coordination between independent clusters of resources, and allows for cloud datacenters to jointly execute applications.

### C. Federated Orchestration

Workload placement within federated clusters requires federated orchestration systems. Such systems, including Borg [29], Twine, [27], Federated Yarn [28] and Hydra [5], are designed to schedule workloads across multiple clusters of machines. As shown in Fig. 2, workload placement is performed by clusters sharing resource consumption either through a centralized state store or gossiping protocols. Such approaches enables federated clusters to scale to tens of thousands of nodes and across several distinct geographic regions whilst presenting a single resource pool to application frameworks. Furthermore the distributed architecture of federated orchestration systems are highly suited resource management across disparate clusters, enabling automated execution of application workflows.

A common design assumption found across these federated orchestration systems is their ability to capture and exploit intra-cluster performance for application placement and execution decisions. However, these systems omit inter-cluster performance in their decision making. This is particularly important in the context of federation given clusters will exhibit heterogeneous operational conditions in terms of network volatility [3], resource contention [30], and scheduling policies

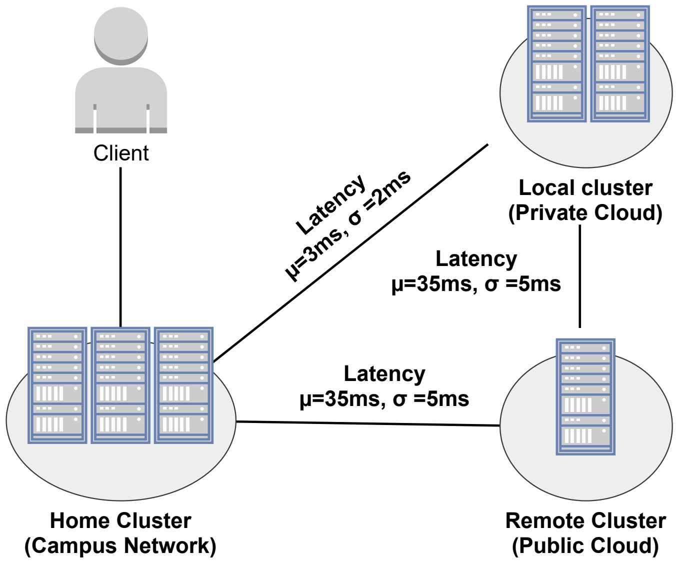


Fig. 3: Federated cluster infrastructure used in experiments.

– conditions inherent to federated clusters (and federated cloud in general) – and all of which are known to negatively affect application performance. Thus omission of inter-cluster performance in federated orchestration can result in sub-optimal application placement across clusters, resulting in application performance degradation.

## III. Related Work

Orchestration systems have been an active area of research within the systems community for decades, evolving from the first cluster computing [24], to Grid computing [9] through to cloud computing [14], [32]. There exist several federated orchestration systems: Borg [29] and Kubernetes Federation[1] enables scheduling application workflows across multiple clusters. Resource orchestration and workload placement is designated to a single master cluster, responsible for executing a centralized scheduling policy managing resource reservation in a distributed Paxos data store. Hydra [5] enables dynamic composition of cluster specific policies. Rather than electing a single master, Hydra offers a decentralized model where worker nodes can request resources from other federated resource manager. In this model each cluster is capable of making local cluster decisions whilst offloading excess resource requests to remote clusters.

Federated orchestration systems capture aggregate resource capacity and reservation, and apply traditional max-min scheduling polices [11], modeling federation resources as a flat hierarchies [5]; thus cannot distinguish inter-cluster performance constraints. Moreover, while there exist application schedulers for federated cluster environments which can capture cluster latency in decision making evaluated through experimentation [8], [13] or simulation [12]. However, these are designed to operate for a single application framework, and are thus not generalizable to other workload types.

[1]https://github.com/kubernetes-sigs/kubefed

## IV. Federated Cluster Orchestration Study

### A. Experiment Setup

**Methodology.** Experiment objectives are two-fold. (1) We empirically demonstrate the importance of capturing inter-cluster performance characteristics within federated orchestration systems, and (2) we perform multiple micro-benchmarks in a federated cluster environment under different operational conditions to demonstrate the performance impact of the following:

- ▷ **Network latency**: A typical cause of workload delay in clusters [4]. Our federation was configured to reflect possible federated clusters conditions; home, local and remote each possessing a third of our federations nodes were configured to reflect dynamic inter-cluster conditions. We imposed network delays and packet loss via leveraging Linux Network Packet Scheduler interface *Traffic Control*. Latency was configured between 0–50ms at increasing 10ms intervals (0ms, 10ms...50ms) with average deviation of [2-4]ms, reflective of modern network latency [20]. A remote subcluster, was configured to exhibit our configured network delays between itself and other clusters in our federation, whilst a local subcluster featured nominal latencies of between [1-5]ms between itself and our home cluster. Such a configuration is representative of a federation composed of a cluster spanning multiple regions.
- ▷ **Application Type:** We selected three application types representative of workloads within federated cluster environments [23]: (1) *Streaming*: Spark DStream [31] using the *WordCount* benchmark configured with a queued input stream, (2) *Machine Learning*: Linear Regression with Stochastic Gradient Descent (SGD) using SparkPerf[2] running 100 iterations; and (3) *Batch*: Apache Hadoop [29] using the *TBC-H* benchmark. Given that Hadoop *Map* and *Reduce* phases exhibit different execution and resource patterns, where appropriate our analysis distinguishes between *Batch-Map* and *Batch-Reduce*.
- ▷ **Contention:** Demonstrated to degrade centralized cluster architecture performance during orchestration [18]. *Linux stress* was configured to inject specified CPU utilization levels. We were interested in exploring high contention federated cluster scenarios (as other experiments intrinsically execute applications within low-medium contention), cluster CPU contention levels were configured at (70%,75%,80%,85%).[3]

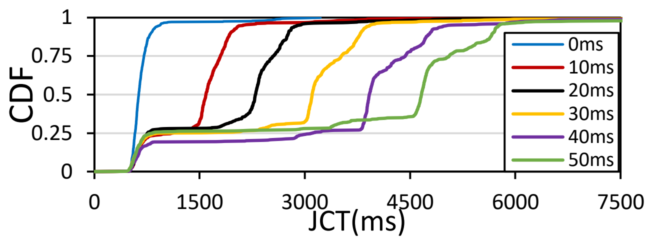
**Environment.** A 30-node federated cluster infrastructure was constructed as shown in Fig. 3, with each node comprising Intel i7-4770 Quad Core @ 3.2Ghz, 8GB RAM, 256GB SSD, 1GBps NIC, running Ubuntu 18.04. *Linux Stress*[4] and *TrafficControl*[5] were used to control CPU utilization and

[2]https://github.com/databricks/spark-perf/blob/master/config/config.py.template

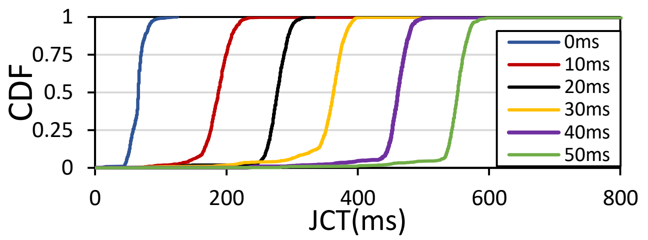
[3]90% and 95% CPU contention resulted in frequent application failures due to dropped heartbeat packets and CPU thrashing.

[4]https://linux.die.net/man/1/stress

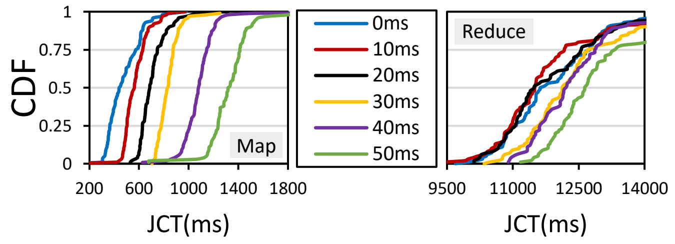
[5]https://linux.die.net/man/8/tc



(a) Streaming

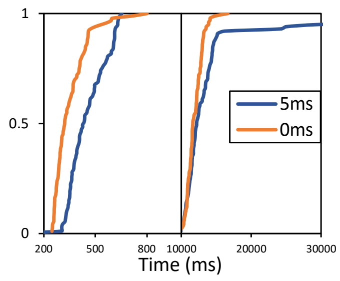


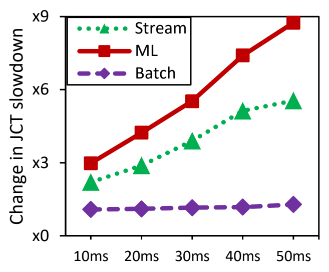
(b) ML



(c) Batch

Fig. 4: Application JCT with inter-cluster latency.





(a) HDFS (b) Application types

Fig. 5: Overview of inter-cluster latency sensitivity.

TABLE I: Application JCT statistics in federated orchestration.

| Latency | Streaming $\mu$ | Streaming $\sigma$ | ML $\mu$ | ML $\sigma$ | Batch $\mu$ | Batch $\sigma$ |
|---|---|---|---|---|---|---|
| 0ms | 702.0 | 335.0 | 65.1 | 11.8 | 6190.9 | 5782.3 |
| 10ms | 1546.8 | 1070.1 | 194.0 | 167.9 | 6693.1 | 7496.7 |
| 20ms | 2027.7 | 1394.3 | 275.5 | 26.4 | 6854.9 | 7542.9 |
| 30ms | 2738.0 | 1592.2 | 359.7 | 131.2 | 7141.6 | 7634.8 |
| 40ms | 3602.0 | 2008.4 | 481.9 | 423.7 | 7295.7 | 7382.4 |
| 50ms | 3894.9 | 2464.1 | 569.1 | 346.3 | 7960.2 | 8740.7 |

network latencies. Nodes were assigned to specific clusters within the federation, interconnected with 10Gbps intra-node bandwidth. Each cluster (and respective nodes) deployed separate instances of Apache Yarn 3.2.1 federation [5], [28]. All experiments contained three cluster types: *Home* hosts the application master or control process, representing a datacenter interacting with client devices. *Local* represents another datacenter with minor network latency from *Home*, and *Remote* representing a cloud datacenter with larger latency between both clusters.

**Metrics.** 900 jobs of each application type were submitted to the federated cluster environment under the operational scenarios described above. The metrics collected were *Job Completion Time* (JCT): end-to-end completion time of a single job, recorded from the start of the job's AM execution and completion upon termination; *Task Duration*: execution of a task in a job, and *Makespan*: end-to-end completion time of executing all jobs. Metrics were collected via Prometheus, parsing application logs, and querying history servers.

## *B. Experiment Results*

**Cluster Latency.** Applications exhibit different JCTs patterns within federated cluster environments as shown in Table I, ranging between 6–569ms (ML), 702–3894ms (Streaming), and 6190–8740ms (Batch) when clusters are exposed to 0–50ms latency, respectively. It is observable that application JCT increases substantially when exposed to inter-cluster latency as shown in Fig. 5b and Table III, resulting in 2.9x (Stream) and 2.2x (ML) slowdown for 10ms latency increase, and in the worst case 5.5x and 8.7x, representing considerable performance degradation of applications within federated orchestration. In contrast, Batch appear unaffected by intra-cluster latency, with 1.28x JCT increase in the worst case scenario. This is because MapReduce tasks are embarrassingly parallel and feature little task inter-dependance. Furthermore, in Fig. 5a We show MapReduce is sensitive to cold storage data locality, by increasing latencies between execution and HDFS data nodes by 5ms, we observed 1.2x slowdown. Thus, Contrasting workloads featuring higher task interdependence and communications overheads such as those found in SGD and Wordcount workloads, posses lower affinities to degraded inter-cluster operational conditions and exhitbit higher levels of JCT slowdown.

Whilst a JCT increase by several hundreds of milliseconds may appear relatively minor, it appears to have considerable performance impact within federated cluster environments. For example, the overall job makespan of ML jobs increases from 26 to 73 minutes for 10ms intra-cluster latency, and to 3.3 hours for 50ms. This is particularly important with the growing prominence of training and inference at the edge [33], where even an additional fraction of a second could be detrimental to sensitive ML-driven applications in security, self-driving vehicles, and traffic management [22].

**Contention.** CPU cluster contention negatively affects the majority of application types, as summarized in Table III. At 85% cluster contention, Batch and Streaming jobs suffered 6.5x and 12.0x performance slowdown when compared to low contention scenarios (i.e. 0ms latency execution in Table I and Fig. 4). Furthermore, as shown in Fig. 6, it is apparent that 70%-80% cluster CPU resulted in increasing left-skewness of JCT distribution for all but one application types. In contrast,

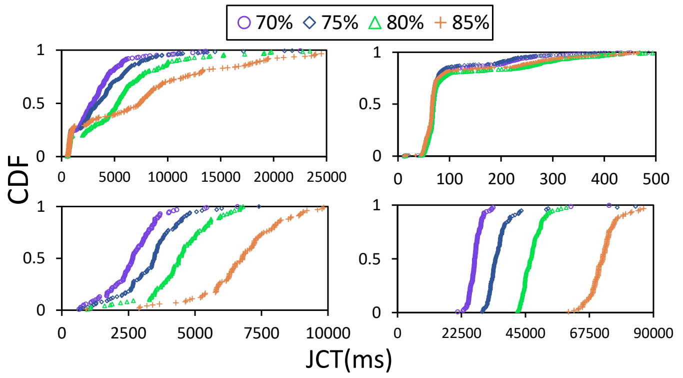


Fig. 6: Application JCT with CPU cluster contention: Streaming *(top left)*, ML *(top right)*, Batch-Map *(lower left)*, Batch-Reduce *(lower right)*.

TABLE II: Application affinity in federated clusters.

| Application | Inter-cluster latency | Contention |
|---|---|---|
| Streaming | Moderate | Moderate |
| ML | Low | High |
| Batch | High | Low (locality) |

ML jobs appear barely affected at higher cluster contention – with only 1.6x slowdown – following near identical JCT patterns in all cluster contention scenarios. This is primarily due to the SGD workload being more data- rather than CPU-intensive.

**Tailing behavior.** Across experiments, we observed applications exhibiting differing tailing behavior. This can be observed in Streaming jobs with high inter-cluster latency (Fig. 4) as well as Streaming and ML jobs exposed to high cluster contention (Fig. 6). Tailing behavior manifests as stragglers, i.e. abnormally slow tasks [6], caused by resource contention, daemon processes, and head-of-line blocking. Stragglers manifested on average in 2.5% (ML), 5% (Streaming), and 8% (Batch) of jobs, reinforcing similar levels in previous studies of production cloud data centers leveraging centralized orchestration systems [10].

**Application diversity.** The reason for different JCT slowdown pattern across applications is due to their architecture model. Streaming jobs partition streaming tasks into sets of stateless, deterministic micro-batches executed at fixed intervals, thus making it moderately susceptible to latency (delayed executor rescheduling per interval) and contention (larger computation requirements). ML jobs are data-intensive and iterative, hence relying on low inter-process latency and memory bandwidth. Batch jobs use a combination of parallel processing (Map) and data consolidation (Reduce), thus both phases are sensitive to CPU contention and dependent on data locality, and thus dependent on network bandwidth rather than latency.

The above experiments indicate that Streaming and Batch-Map application performance are particularly sensitive to higher cluster CPU contention, whereas ML applications appears to be minimally affected.

TABLE III: JCT increase from cluster performance conditions.

| | Streaming | ML | Batch |
|---|---|---|---|
| Inter-cluster latency | 5.5x | 8.7x | 1.28x |
| Cluster contention | 12.0x | 1.6x | 6.5x |

## V. Evaluation of Findings from Empirical Study

Our experiments discussed in §IV-B has uncovered several insights into the behavior of applications executing within federated cluster environments, as well as highlighted current limitations within existing federated orchestration systems:

**Minor inter-cluster latency between federated cluster applications causes severe performance degradation.** A modest network latency of 10ms (commonly found in close-range WiFi environments [25]) results in a 2.2x and 2.9x application slowdown in Streaming and ML applications. This indicates that a large portion of existing application frameworks (e.g. *Apache Samza* [34], *Flume* [15], *Storm* [16]) may inherently be unsuitable to effectively operate in federated cluster environments – even when using current federated orchestration systems – without significant modification. These findings are reinforced by studies on intra-cluster latency increases between *2-1000$\mu$* seconds latency results in considerable workload degradation [21], which are order of magnitudes lower than latencies found in federated cluster environments. Whilst some application schedulers capture inter-cluster performance [3], [18], they are restricted to a single application framework, debilitating their generalizability to facilitate increased workload diversity necessary to ensure federated cluster adoption.

**Insight 1.** *Federated orchestration require mechanisms capable of sharing inter-cluster performance metrics allowing for localized tracking of cluster saturation and inter-cluster network volatility for federated scheduling policies.*

**Federated orchestration can be improved by using inter-cluster performance in decision making.** Whilst the federated orchestration system deployed within the federated cluster infrastructure was able to successfully execute all submitted jobs, our experiments have shown that such systems are unable to effectively deal with inter-cluster performance degradation (stemming from latency and resource contention). As discussed in §III, this design assumption is shared across all existing federated orchestration systems capable of supporting generalized workload deployment and execution [5], [29], and the few that do consider resource reservation [5], [11], [29] and omit contention and latency in decision making causing degraded performance from low quality placement.

**Insight 2.** *Federated orchestration scheduling requires policies capable of profiling, modeling, and exploiting cluster resource contention and network latency when placing applications.*

**Application architectures are key performance indicators when orchestrating workloads in federated environments.** We studied three different workloads: Worcount (*streaming*) TBC-H (*batch*) and SGD (*Machine Learning*) to demonstrate how different distributed application architectures are impacted by inter-cluster operational conditions. Such workload may be leveraged to create a 'profile' to describe the expected performance of an application when deployed within a federated cluster. Capturing the full diversity of such profiles for different application types requires measuring and modelling application performance at run-time, thus improving scheduling decisions iteratively over time.

**Insight 3.** *Federated orchestration policies should account for applications architectures when when considering scheduling policies which should be applied for an application.*

**Federated cluster applications exhibit unique performance affinities for different operational conditions.** Workload affinity describes a workloads relative performance **w.r.t** operational conditions (*e.g. inter-cluster latencies and localised resource contention*). From analyzing application performance profiles when exposed to cluster latency and cluster resource contention, we found that applications using different communication models appear to exhibit different levels of sensitivity to JCT slowdown stemming from inter-cluster performance characteristics as summarized in Table II and Fig. 5. Streaming has high contention sensitivity (12x) and moderate latency sensitivity (2.2–5.5x). ML has low contention sensitivity (1.6x) and moderate latency sensitivity (2.2–8.7x). Batch has moderate cluster contention (6.5x) and latency tolerance dependent on data locality. Whilst the impact of cluster resource contention upon orchestration has been studied within the context of centralized architectures [11], to our knowledge our analysis is among the first works to empirically demonstrate the severity of how both contention and inter-cluster latency debilitates the performance of federated cluster application execution.

**Insight 4.** *Federated orchestration requires enhanced scheduling policies that classify and exploit application workload resource usage, communication models, and task interdependence in federated cluster environments.*

**Straggler manifestation is a considerable challenge in federated cluster computing.** A finding that was not considered within our original objective was the identification of straggler phenomena in experiments, whose impact upon application performance appears to be worsen in federated federated cluster environments. Current approaches to address stragglers in centralized architectures rely on launching speculative executors for mitigation, however such approaches still result in 47% JCT increase [1], and will be made worse considering the context of interactive applications that require federated cluster capability. Hence, as federated cluster scale (in terms of nodes, clusters, tasks) increases so does the probability of straggler manifestation. Placement of speculative containers must consider workload classification to avoid placement into a clusters that may perform worse than the straggler task. This is a problem as clusters are exposed to localized and temporal resource utilization and network conditions requiring global monitoring of cluster constraints.

**Insight 5.** *New straggler detection and mitigation strategies need to be designed that capture inter-cluster performance volatility when launching speculative tasks.*

## VI. Conclusions

In this paper we have conducted an empirical study of the challenges faced by federated cluster orchestration. Through experiments we have demonstrated how federated cluster applications are vulnerable to detrimental inter-cluster performance conditions resulting in 8.7x-12x slowdown that current federated orchestrations systems are unable to overcome. Our study has uncovered four key findings - including the severity of performance degradation from minor cluster latency, different workload performance affinities in federated cluster environments, and the problems of inter-cluster straggler manifestation. Informed by these findings, there are several research directions to pursue to address identified challenges. We hope that the findings and insights provided within the study will aid the federated cluster and cloud computing communities towards development of new federated scheduling policies capable of capturing workload sensitivity to inter-cluster performance metrics.

## Acknowledgements

This work is supported by the UK EPSRC (EP/P031617/1). We thank Damian Borowiec for their help in reviewing this paper.

## References

[1] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: Attack of the clones. In *Proceedings of the 10th Symposium on Networked Systems Design and Implementation (NSDI)*, pages 185–198, 2013.

[2] Luiz André Barroso and Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2009.

[3] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti, and Matteo Nardelli. On QoS-Aware scheduling of data stream applications over fog computing infrastructures. In *Proceedings of the IEEE Symposium on Computers and Communications (ISCC)*, pages 271–276, 2016.

[4] Wei Chen, Aidi Pi, Shaoqi Wang, and Xiaobo Zhou. Characterizing scheduling delay for low-latency data analytics workloads. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 630–639, 2018.

[5] Carlo Curino, Subru Krishnan, Konstantinos Karanasos, Sriram Rao, Giovanni M. Fumarola, Botong Huang, Kishore Chaliparambil, Arun Suresh, Young Chen, Solom Heddaya, Roni Burd, Sarvesh Sakalanaga, Chris Douglas, Bill Ramsey, and Raghu Ramakrishnan. Hydra: A federated resource manager for data-center scale analytics. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, page 177–191, 2019.

[6] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56:74–80, 2013.

[7] Pamela Delgado, Florin Dinu, Anne-Marie Kermarrec, and Willy Zwaenepoel. Hawk: Hybrid Datacenter Scheduling. In *Proceedings of the USENIX Conference on Usenix Annual Technical Conference (ATC)*, pages 499–510, 2015.

[8] Robert Evans. Apache Storm, a hands on tutorial. In *Proceedings of the International Conference on Cloud Engineering*, IC2E, 2015.

[9] Ian T. Foster and Carl Kesselman. Globus: a metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications*, 11:115 – 128, 1997.

[10] P. Garraghan, X. Ouyang, R. Yang, D. McKee, and J. Xu. Straggler root-cause and impact analysis for massive-scale virtualized cloud datacenters. *IEEE Transactions on Services Computing*, 12(1):91–104, 2019.

[11] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant Resource Fairness: Fair allocation of multiple resource types. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, page 323–336, 2011.

[12] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. *CoRR*, abs/1606.02007, 2016.

[13] Karim Habak, Ellen W. Zegura, Mostafa Ammar, and Khaled A. Harras. Workload management for dynamic mobile device clusters in edge Femtoclouds. In *IEEE/ACM Symposium on Edge Computing (SEC)*, New York, NY, USA, 2017. Association for Computing Machinery.

[14] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, page 295–308, USA, 2011. USENIX Association.

[15] Steve Hoffman. *Apache Flume: distributed log collection for Hadoop*. Packt Publishing Ltd, 2013.

[16] Muhammad Hussain Iqbal and Tariq Rahim Soomro. Big Data Analysis: Apache Storm Perspective. *International Journal of Computer Trends and Technology*, 19(1):9–14, 2015.

[17] Yuxuan Jiang, Zhe Huang, and Danny H. K. Tsang. Challenges and solutions in fog computing orchestration. *IEEE Network*, 32(3):122–129, 2018.

[18] Shweta Khare, Kaiwen Zhang, Hongyang Sun, Aniruddha Gokhale, Julien Gascon-Samson, Yogesh Barve, Anirban Bhattacharjee, and Xenofon Koutsoukos. Linearize, predict and place: Minimizing the Makespan for edge-based stream processing of directed acyclic graphs. *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, SEC 2019*, pages 1–14, 2019.

[19] Lars Larsson, Harald Gustafsson, Cristian Klein, and Erik Elmroth. Decentralized Kubernetes Federation Control Plane. pages 354–359, 2020.

[20] Diana A. Popescu and Andrew W. Moore. PTPmesh: Data center network latency measurements using ptp. In *25th Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 73–79, 2017.

[21] Diana Andreea Popescu and Andrew W. Moore. No Delay: Latency-Driven, Application Performance-Aware, Cluster Scheduling. 2019.

[22] Vicent Sanz Marco, Ben Taylor, Zheng Wang, and Yehia Elkhatib. Optimizing deep learning inference on embedded systems through adaptive model selection. *Transactions on Embedded Computing Systems*, 19(1), February 2020.

[23] Simar Preet Singh, Anand Nayyar, Rajesh Kumar, and Anju Sharma. Fog computing: from architecture to edge computing and big data processing. *The Journal of Supercomputing*, 75(4):2070–2105, 2019.

[24] Thomas Sterling, Donald J. Becker, Daniel Savarese, John E. Dorband, Udaya A. Ranawake, and Charles V. Packer. Beowulf: A parallel workstation for scientific computation. In *Proceedings of the 24th International Conference on Parallel Processing*, pages 11–14, 1995.

[25] Kaixin Sui, Mengyu Zhou, Dapeng Liu, Minghua Ma, Dan Pei, Youjian Zhao, Zimu Li, and Thomas Moscibroda. Characterizing and improving WiFi latency in large-scale operational networks. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys, page 347–360. ACM, 2016.

[26] X. Sun, C. Hu, R. Yang, P. Garraghan, T. Wo, J. Xu, J. Zhu, and C. Li. Rose: Cluster resource scheduling via speculative over-subscription. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 949–960, 2018.

[27] Chunqiang Tang, Kenny Yu, Kaushik Veeraraghavan, Jonathan Kaldor, Scott Michelson, Thawan Kooburat, Aravind Anbudurai, Matthew Clark, Kabir Gogia, Long Cheng, Ben Christensen, Alex Gartrell, Maxim Khutornenko, Sachin Kulkarni, Marcin Pawlowski, Tuomas Pelkonen, Andre Rodrigues, Rounak Tibrewal, Vaishnavi Venkatesan, and Peter Zhang. Twine: A unified cluster management system for shared infrastructure. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 787–803. USENIX Association, November 2020.

[28] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddarth Seth, Bikas Saha, Carlo Curino, Owen OMalley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache Hadoop YARN: yet another resource negotiator. *Proceedings of the 4th annual Symposium on Cloud Computing (SOCC)*, pages 1–16, 2013.

[29] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. *Proceedings of the 10th European Conference on Computer Systems (EuroSys)*, pages 1–17, 2015.

[30] Renyu Yang, Chunming Hu, Xiaoyang Sun, Peter Garraghan, Tianyu Wo, Zhenyu Wen, Hao Peng, Jie Xu, and Chao Li. Performance-aware Speculative Resource Oversubscription for Large-scale Clusters. *IEEE Transactions on Parallel and Distributed Systems*, 31(7):1499–1517, 2020.

[31] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. Discretized streams: Fault-tolerant streaming computation at scale. *SOSP 2013 - Proceedings of the 24th ACM Symposium on Operating Systems Principles*, (1):423–438, 2013.

[32] Zhuo Zhang, Chao Li, Yangyu Tao, Renyu Yang, Hong Tang, and Jie Xu. Fuxi: A fault-tolerant resource management and job scheduling system at internet scale. *Proc. VLDB Endow.*, 7(13):1393–1404, August 2014.

[33] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019.

[34] Zhenyun Zhuang, Tao Feng, Yi Pan, Haricharan Ramachandra, and Badri Sridharan. Effective multi-stream joining in apache samza framework. *Proceedings - 2016 IEEE International Congress on Big Data, BigData Congress 2016*, pages 267–274, 2016.