Cost-Aware Predictive Planning for Decentralized Serverless Workflow Execution

Diogo Jesus, Luís Veiga

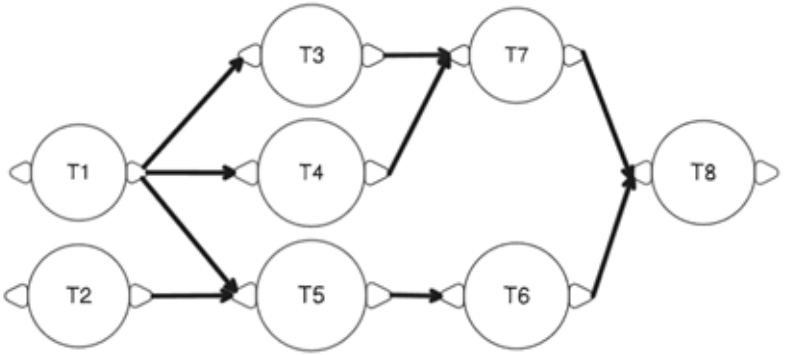
GECON 2026 - 21th International Conference on the Economics of Grids, Clouds, Systems, and Services
Pisa, Italy, 2026/09/10

Motivation

- Serverless Computing
  - **Automatic scaling** (based on demand)
  - **Easier** to configure and deploy
  - **Pay-per-use pricing model**
- Great for **IoT**, **Web Apps**, and **Workflows** (e.g., ETL jobs, e-commerce, machine learning, scientific computing)
- **Serverless Services:** FaaS, BaaS, CaaS, Storage (Databases and Queues)
- **Commercial offerings** (*FaaS*): AWS Lambda, Azure Functions, Google Cloud Functions

Serverless Shortcomings

- Platform Limitations
  - **cold starts**
  - **no direct function communication** (forcing apps to use external services)
  - vendor lock-in
  - monitoring
  - debugging
  - limited execution time
- **Workflows** are particularly affected
- Research projects try to combat these limitations
  - *FaaS* platform extensions
  - Innovative scheduling strategies



DAG representation of a workflow

Contributions

- Analysis of the current serverless landscape
- Implementation of a novel **decentralized serverless workflow execution engine with predictive planning**
- **Evaluation** of the proposed solution, comparing it to WUKONG (Carver B. et al., 2020)

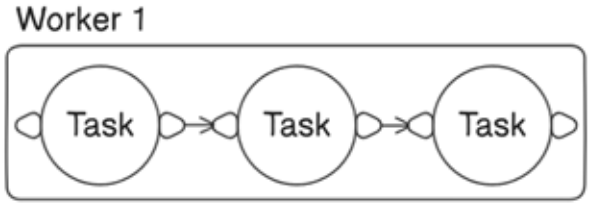
Presentation Roadmap

- Related Work
- Solution
- Evaluation
- Conclusion

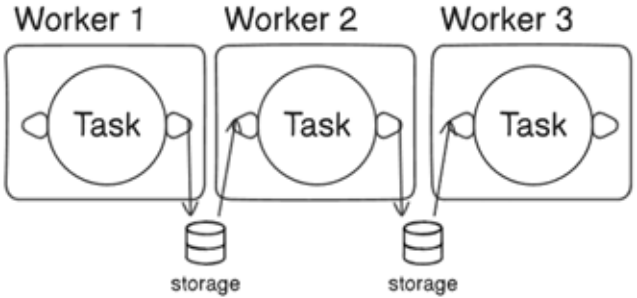
Related Work - Key Concepts

- **Data Locality**: how close the Tasks execute in relation to their dependencies
- **Resource Contention:** when tasks compete for the same resources

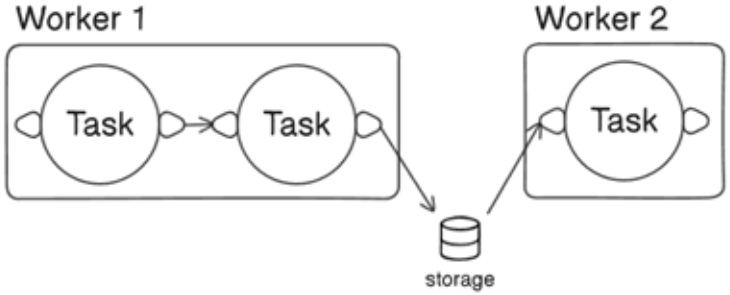
MAX Data Locality
MAX Resource Contention



NO Data Locality
NO Resource Contention



BALANCED Data Locality and Resource Contention

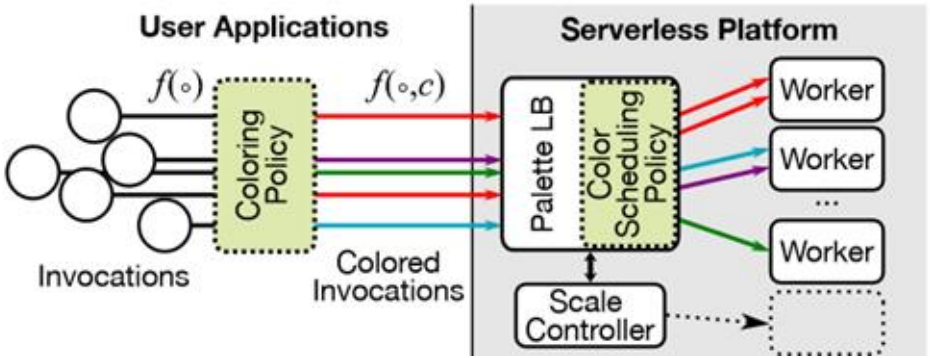


Related Work

Improving Data Locality at the Platform Level

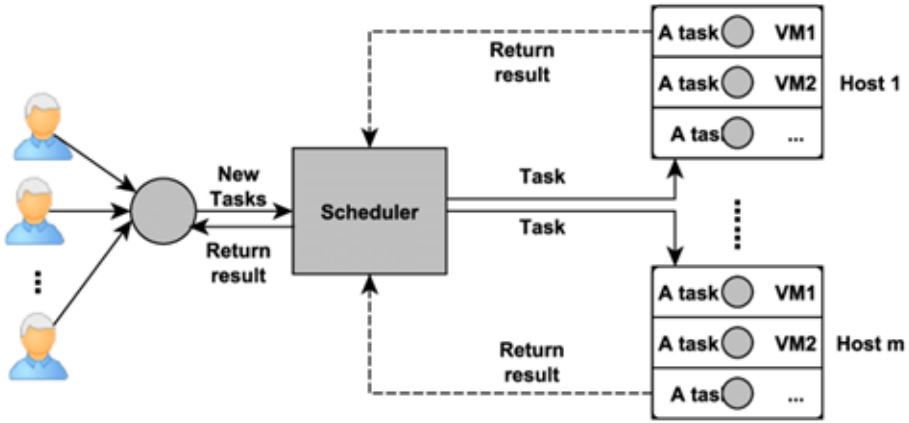
Serverless platform extensions (require modifications to the *FaaS* platform):

- **Palette Load Balancing** (M. Abdi et al., 2023)
  - ***"color"* hints** provided by the developer (some control over locality)
  - same color = same worker



- **Faa$T** (F. Romero et al., 2021)
  - transparent, auto-scaling distributed cache for serverless applications

Related Work - Workflow Scheduling



- **Traditional *IaaS* Schedulers**: Apache Spark, Apache Hadoop, HTCondor
  - hard to manage and use
  - limited scalability
- **Cloud-Native Schedulers**: Apache AirFlow, Prefect, Dagster, Argo Workflows
  - centralized scheduling
- **Stateful Functions**: AWS Step Functions, Azure Durable Functions, Google Cloud Workflows
  - layer of orchestration and state management to stateless functions
  - extra costs
  - vendor-specific workflow languages

Related Work

Research on Serverless Workflow Scheduling

- Leverage **stateless functions** to run workflows without modifying the platform

System	Execution Model	Scheduling Model	Data Locality	Resource Allocation	Scalability
DEWE v3 (Q. Jiang et al. 2017)	**IaaS + FaaS**	Centralized (dedicated)	Medium	Different for short/long tasks	Medium
PyWren (E. Jonas et al., 2017)	FaaS	Centralized (client machine)	Low	Uniform	Medium
Unum (D. H. Liu et al., 2023)	FaaS	**Decentralized** (worker driven)	Low	Uniform	High
WUKONG (Carver B. et al., 2020)	FaaS	**Decentralized** (worker driven)	**High** (Task Clustering and Delayed I/O)	Uniform	High

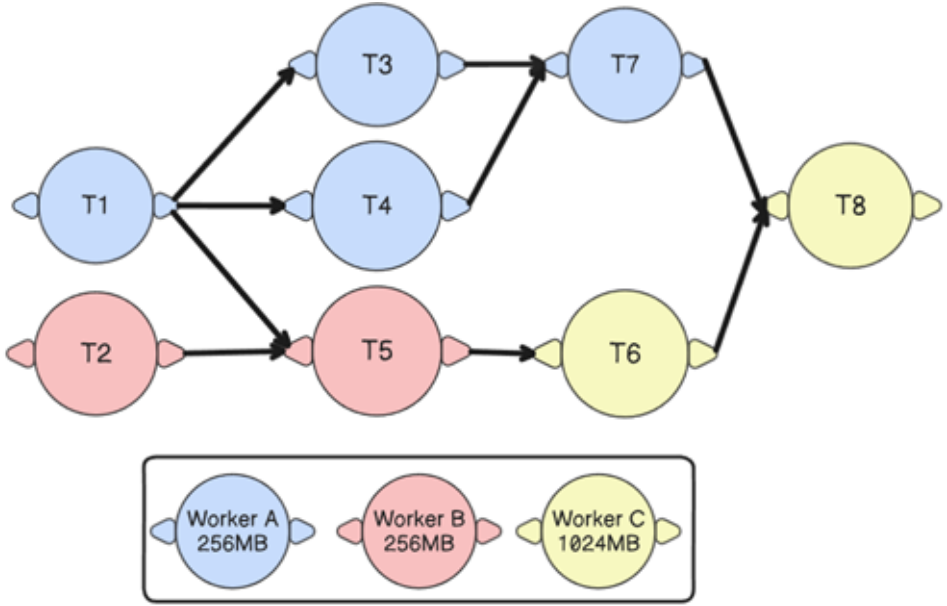
Related Work - Limitations

- **WUKONG**
  - **Optimizations** can hurt performance in certain scenarios
  - Launches **lots of workers** (*N-1* on fan-outs)
- Most scheduling solutions
  - **Same resource configuration** for all tasks
  - **Reactive scheduling**, not leveraging the global knowledge of the workflow
  - Depend on a **Central Entity**
  - Rely solely on heuristics instead of **adapting/learning** from previous workflow runs

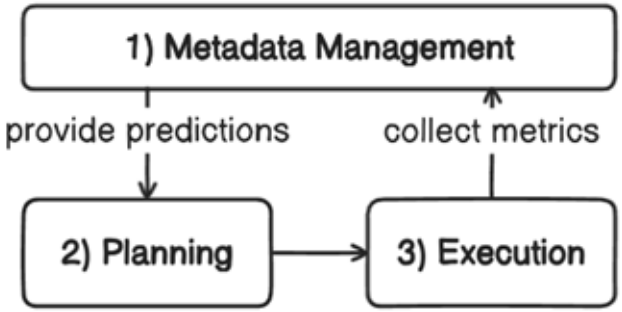
Solution Proposal

A modular **decentralized serverless workflow execution engine**

- Adaptive and Predictive approach (rather than step-by-step/reactive scheduling)
- Use workflow history to create workflow plans (as shown in the image)
- Worker resources adapted to the needs of different Tasks

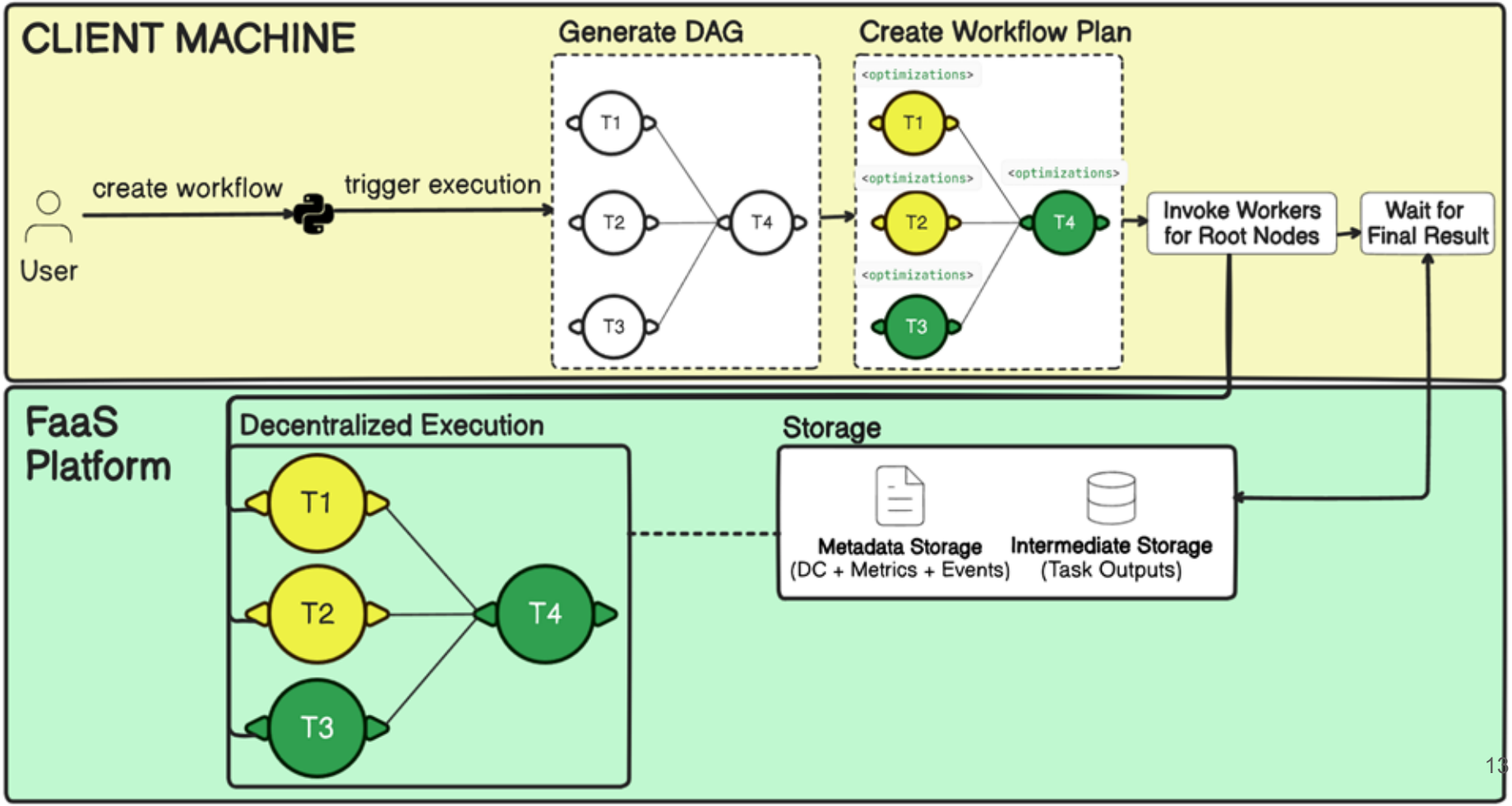


Solution Layers



1. **Metadata Management**
   - Collect metrics
   - Make **predictions** (execution time, data transfer times, tasks I/O, worker startup time and state)
   - Provide a **Workflow Simulation API**
2. **Static Workflow Planning**
   - Produce a workflow plan (Task ➛ Virtual Worker ID + Optimizations assignments)
3. **Decentralized Workflow Execution**
   - *FaaS* workers **follow the plan** in a decentralized manner
   - Use **External Storage** for synchronization and task outputs

Distributed Architecture

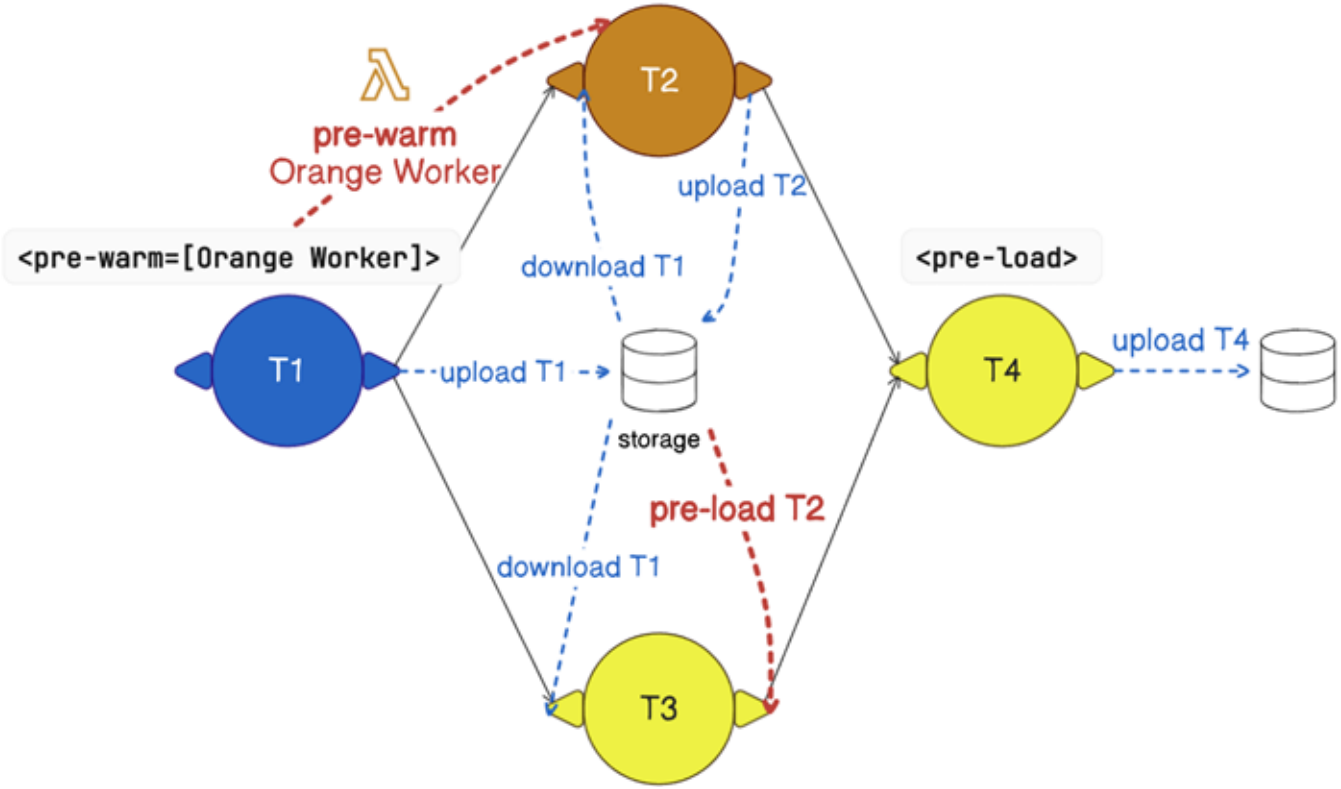


Workflow Planning

- Planning Algorithms
  - **UNIFORM**
    1) **Worker Assignment Algorithm +** same resources to all tasks
  - **NON-UNIFORM**
    2) **Worker Assignment Algorithm +** best resources to all tasks
    3) **Resource Downgrading** algorithm for workers outside the Critical Path
- Optimizations
  - **Pre-Warm:** mitigate cold-start delays
  - **Pre-Load**: download task dependencies as soon as they become available

Optimizations in Action

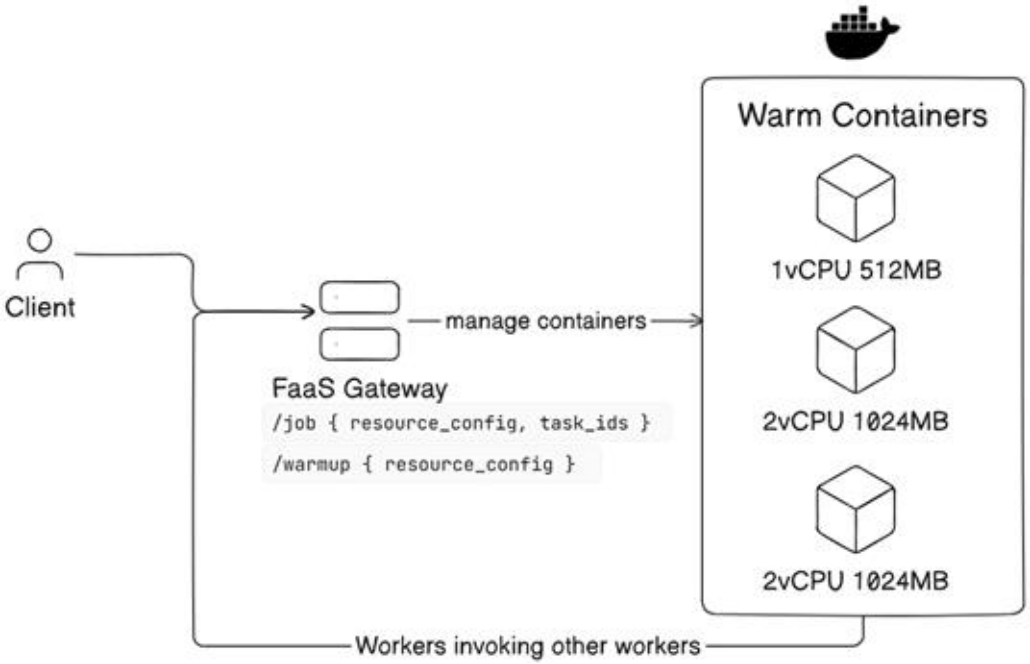
- 3 **Workers**: Blue, Orange, and Yellow
- 2 **Optimizations**: Pre-Warm on Blue Worker and Pre-Load on T4



Evaluation

- **RQs**
  - How **accurate and impactful are our predictions** on serverless workflows?
  - How do our **workflow planning algorithms** compare to WUKONG?
  - How effective are ***Pre-Load*** and ***Pre-Warm*** optimizations?
  - What is the trade-off between performance and cost (**Non-Uniform** vs **Uniform**)?
- **Metrics Collected**
  - Per **task**: exec. time, start time, data transfer times, time waiting for dependencies
  - Per **workflow**: plan, submission time, run-time and resource cost (in GB-seconds)
  - Per **worker**: startup time and state (*warm* or *cold*)

Test Environment - FaaS Emulator



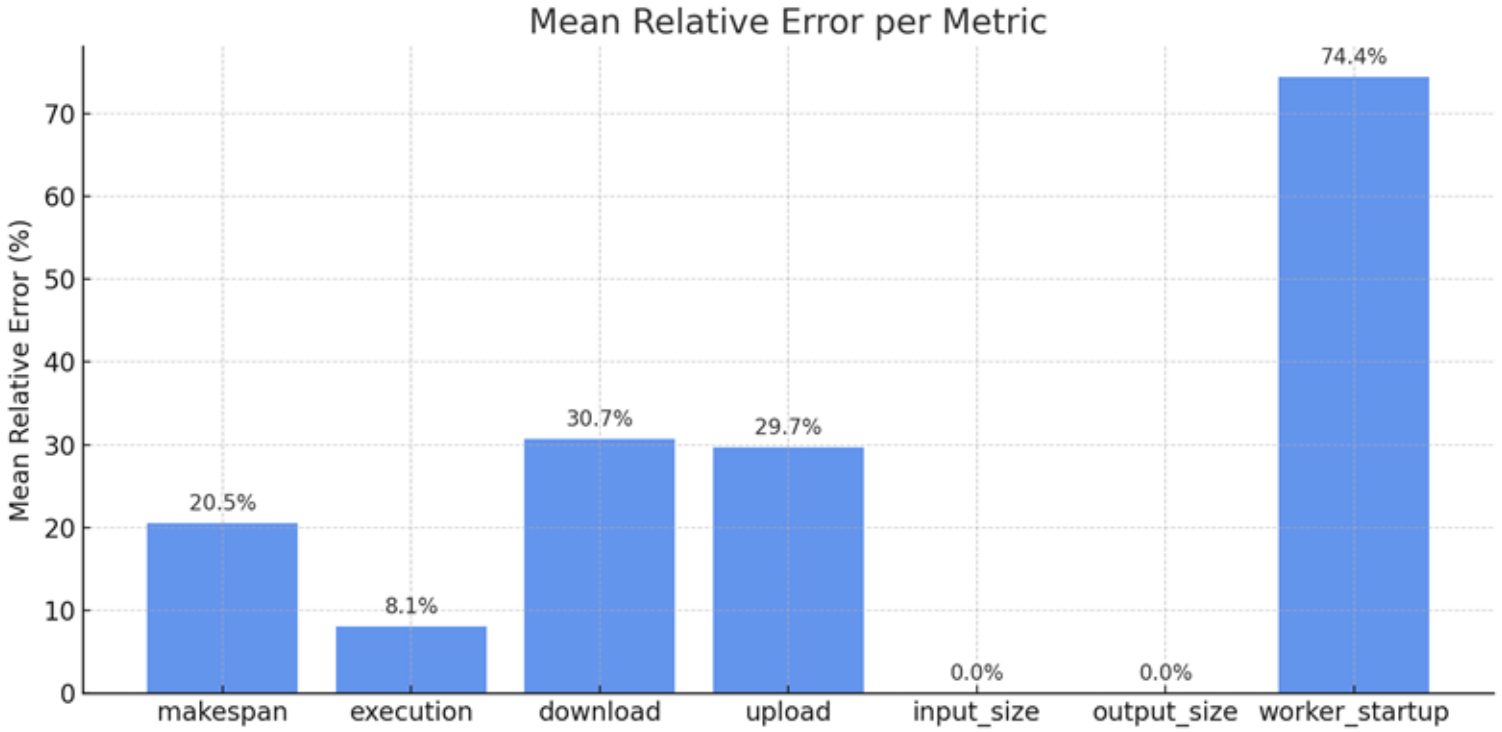
- Semi-predictable environment for testing
- Lower experimental costs

- Each container runs a **single worker's** logic at a time
- Containers have **limited resources**
- Has **Cold Starts**
- ***"Warmup"*** requests launch new containers

Evaluation - Deployment

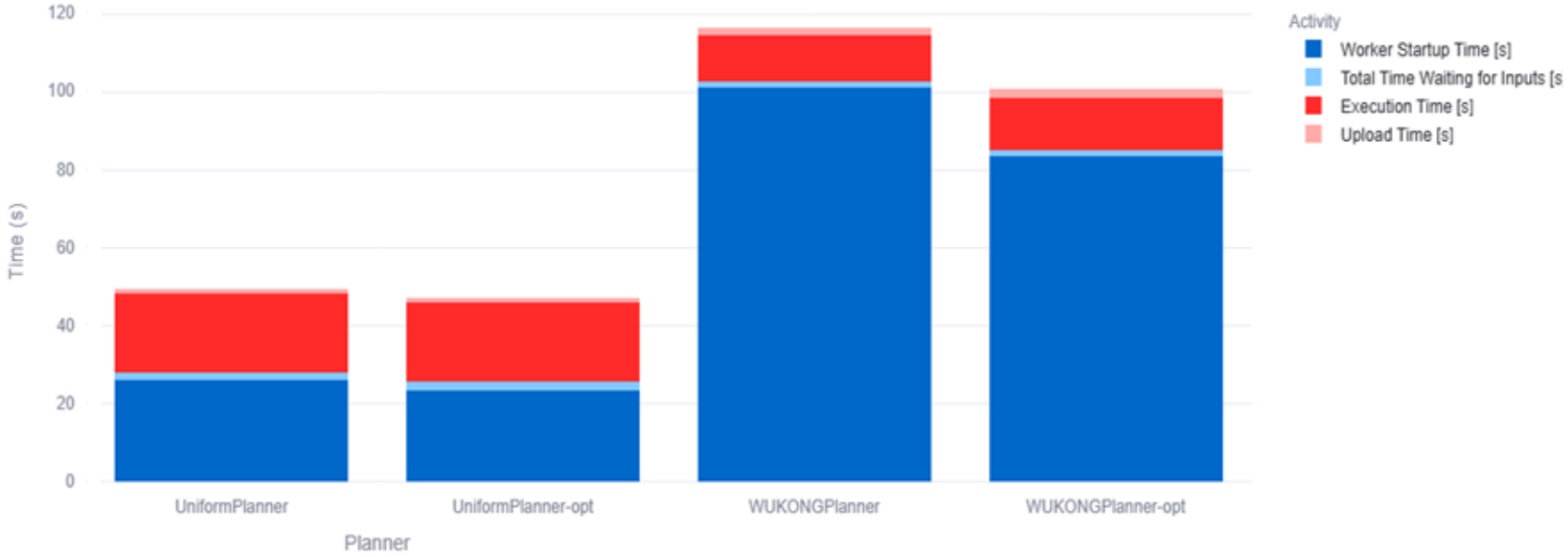
- AMD EPYC 7282 16-Core (32 logical threads, via AMD SMT) processor with 125GiB of RAM, running Ubuntu 22.04.5 LTS
  - ***FaaS* Emulator** (Gateway + Docker Containers)
  - **Intermediate Storage** and **Metadata Storage** (Redis on Docker)
  - **Client** (submitting workflows)
- Tested Setup:
  - 4 **workflows**: Matrix Multiplication, Tree Reduction, Text Analysis, and Image Transformation
  - 6 **planners**: Uniform, Non-Uniform, WUKONG, and optimized variants
  - 3 **SLAs**: Median (P50 = 50th percentile), P75, P90
  - 10 instances of each, leading to a total of 720 workflow instances
- **Uniform** and **WUKONG** workers were limited to **2048 MB RAM**
- **Non-Uniform** workers could use **2048 MB**, **4096 MB**, or **8192 MB**

Results - Predictions Accuracy



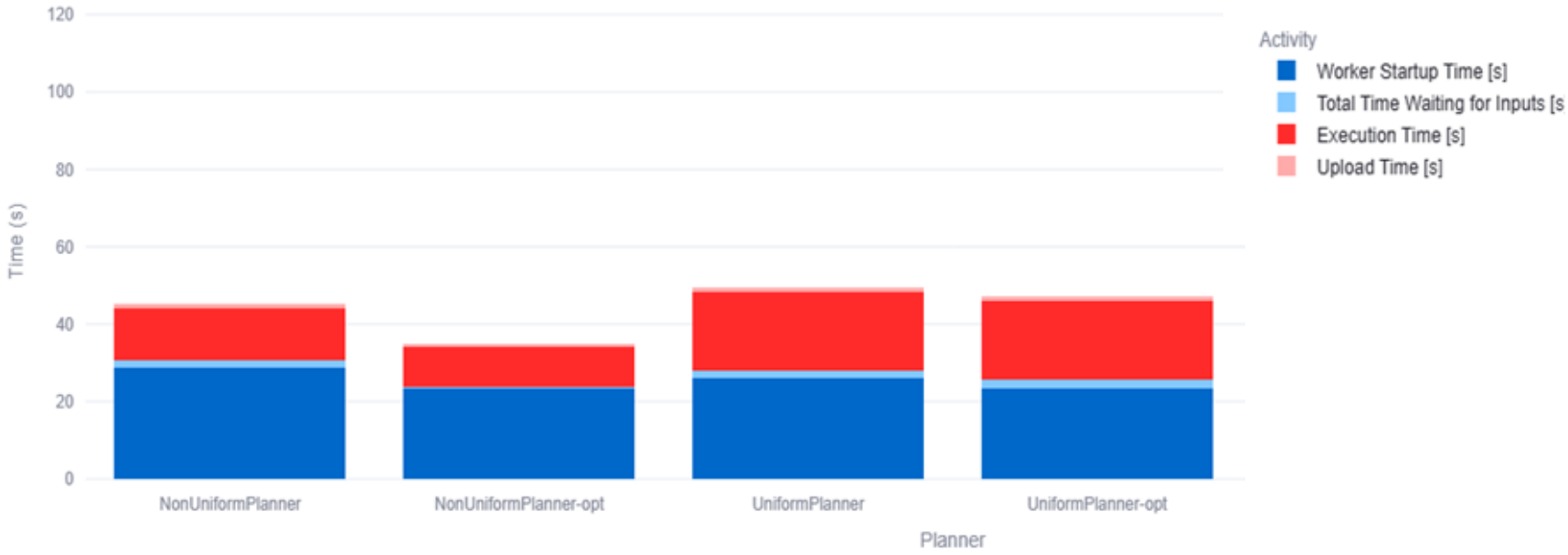
- Decent overall prediction accuracy
- More sophisticated prediction methods could be used to improve accuracy

Results - Time Breakdown (WUKONG + Uniform)



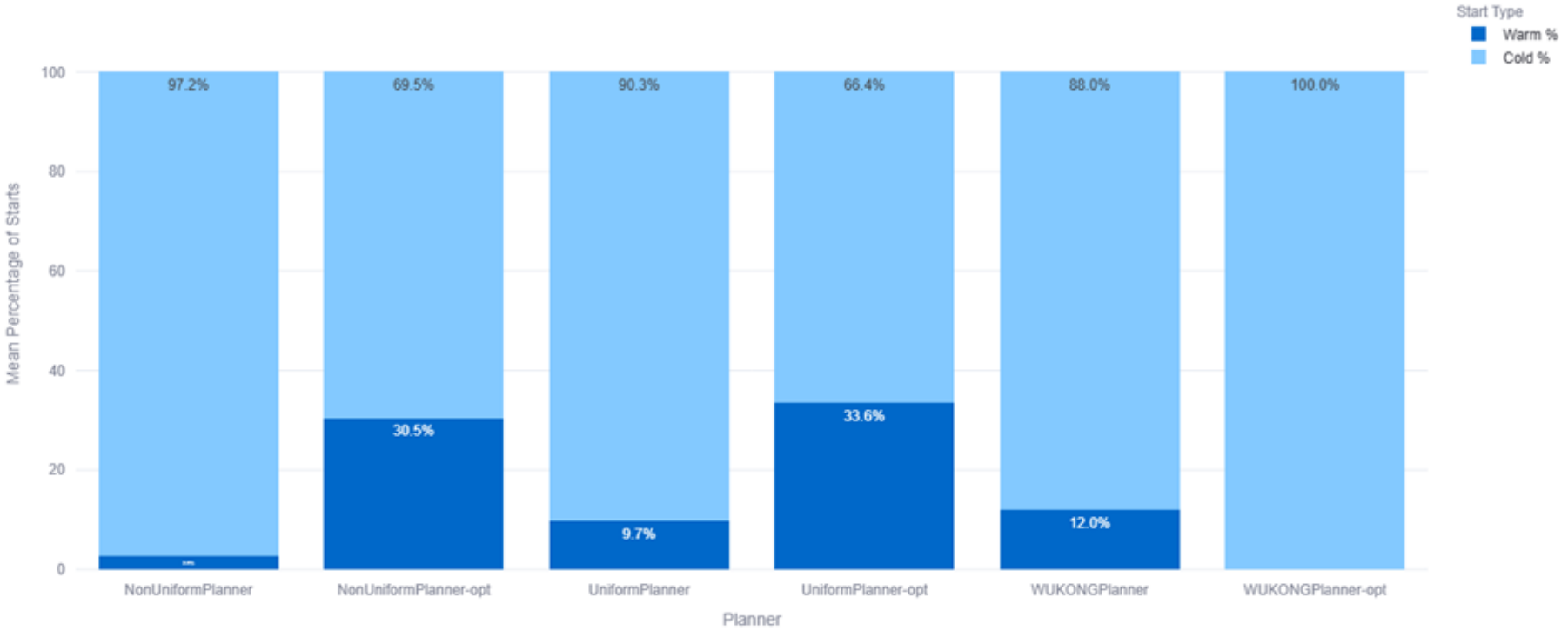
- **WUKONG** uses more workers, leading to greater “worker startup times”
- **Uniform** task execution times are higher due to resource contention (multiple tasks executing concurrently)

Results - Time Breakdown (Non-Uniform and Uniform)



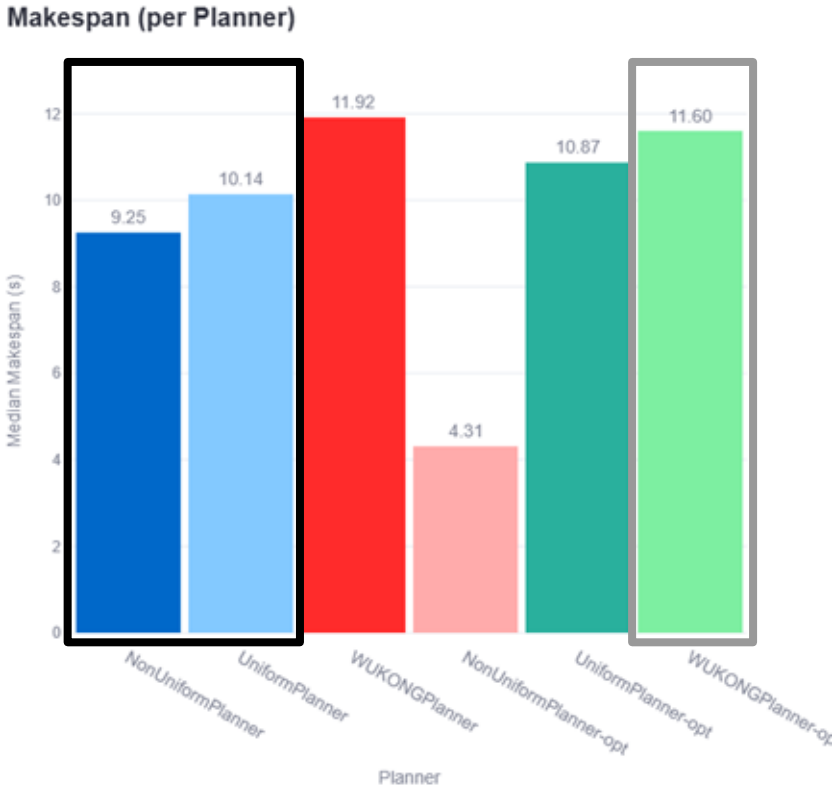
- **Pre-Load** impact: visible in **Non-Uniform** planner, but not in **Uniform** (we think it's due to resource contention)
- **Pre-Warm** impact: less worker startup time, saving up to 6 seconds

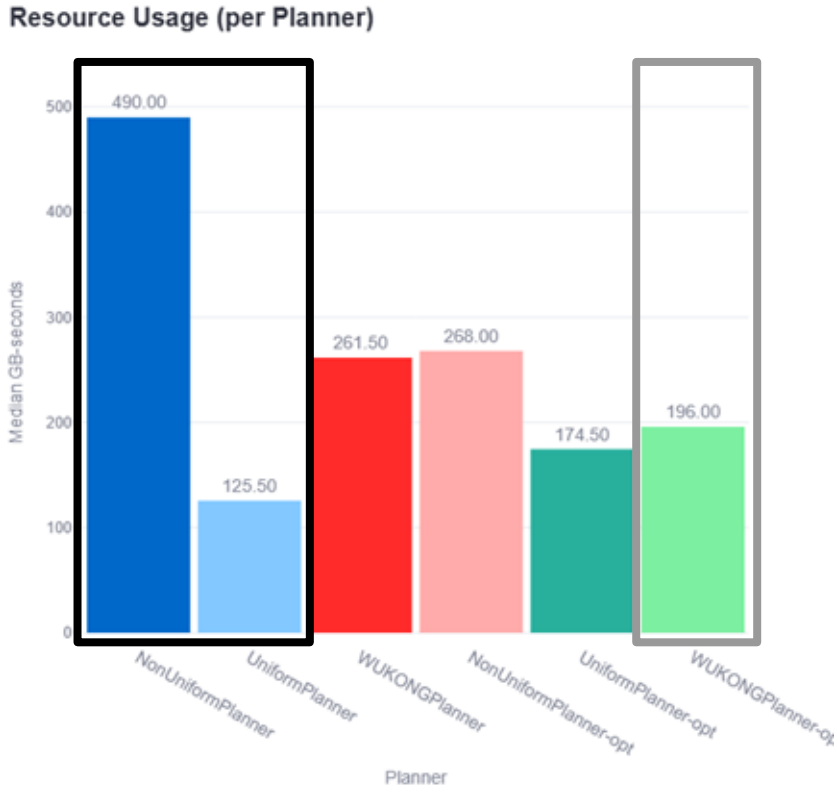
Results - Warm Starts vs Cold Starts



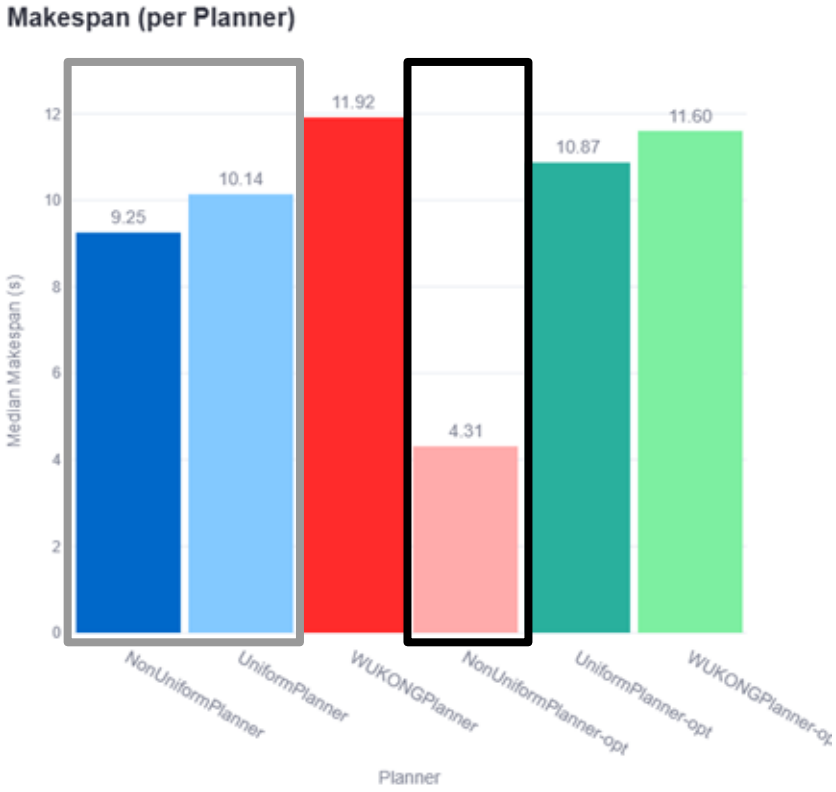
- **Pre-Warm** eliminated **25%** of **Cold Starts**

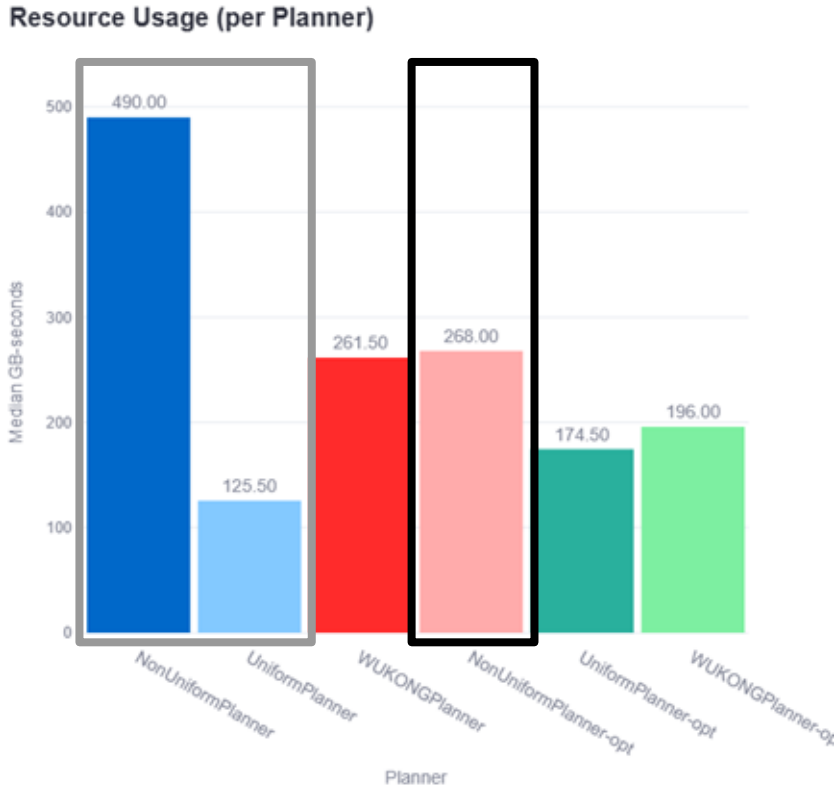
Note: Pre-Warm **doesn't apply to the initial workers**





- **Uniform** vs **WUKONG Optimized:** 12.6% faster, 36% cheaper
- **Non-Uniform** vs **Uniform**: 8.8% faster, 290.4% most costly





- **Non-Uniform Optimized** vs **Non-Uniform**: 53.4% faster, 45.3% cheaper
- **Non-Uniform Optimized** vs **Uniform**: 57.5% (**2.3x**) faster, 113.5% (**2.1x**) more costly

Result Analysis

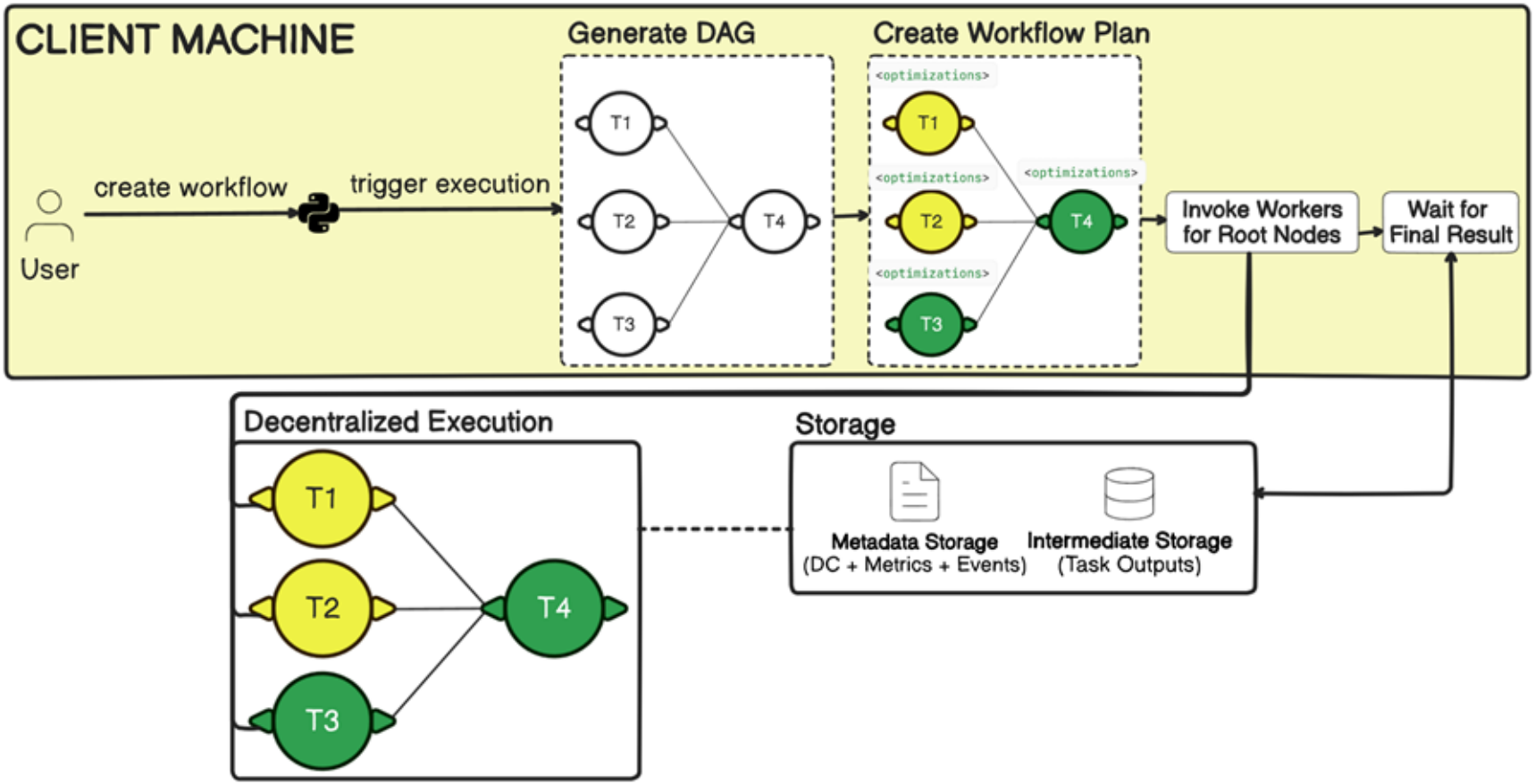
- **Uniform** planner **outperformed WUKONG** (even without optimizations)
  - Faster and Cheaper
  - Demonstrated the importance of balancing data locality and resource contention
- **Optimizations** were very effective for the **Non-Uniform** planner
  - ***2.3x*** faster for ***2.1x*** the cost

Conclusion

- **Achievements**
  - Demonstrated the advantages of an **adaptive prediction-based approach** for serverless workflows
  - Provided a **versatile framework for researchers**
    - scheduling algorithms
    - prediction techniques
    - optimizations
- **Future Work**
  - **Experiment on a commercial platform** (e.g., AWS Lambda): more resources, but more unpredictable
  - **Dynamic Predictive Scheduling**: Allow workers to make predictions “on the fly”

Thank you for your attention

Questions?



References

- https://www.grandviewresearch.com/industry-analysis/serverless-computing-market-report
- Aws lambda. [Online]. Available: https://aws.amazon.com/pt/lambda/
- Azure functions. [Online]. Available: https://azure.microsoft.com/en-us/products/functions
- Google cloud run functions. [Online]. Available: https://cloud.google.com/functions
- B. Carver, J. Zhang, A. Wang, A. Anwar, P. Wu, and Y. Cheng, "Wukong: A scalable and locality-enhanced framework for serverless parallel computing," in *Proceedings of the 11th ACM symposium on cloud computing*, 2020, pp. 1-15. [Online]. Available: https://arxiv.org/abs/2010.07268
- Apache openwhisk. [Online]. Available: https://openwhisk.apache.org/
- Openfaas. [Online]. Available: https://www.openfaas.com/
- Knative. [Online]. Available: https://knative.dev/docs/
- M. Abdi, S. Ginzburg, C. Lin, J. M. Faleiro, I. Goiri, G. I. Chaudhry, R. Bianchini, D. S. Berger, and R. Fonseca, "Palette load balancing: Locality hints for serverless functions," in *EuroSys*. ACM, May 2023. [Online]. Available: https://www.microsoft.com/en-us/research/publication/palette-load-balancing-locality-hints-for-serverless-functions/
- F. Romero, G. I. Chaudhry, I. n. Goiri, P. Gopa, P. Batum, N. J. Yadwadkar, R. Fonseca, C. Kozyrakis, and R. Bianchini, "Faa$t: A transparent auto-scaling cache for serverless applications," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '21. 2021, p. 122-137. [Online]. Available: https://doi.org/10.1145/3472883.3486974
- Y. Tang and J. Yang, "Lambdata: Optimizing serverless computing by making data intents explicit," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 2020, pp. 294-303. [Online]. Available: https://ieeexplore.ieee.org/document/9253018
- S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein, "Encoding, fast and slow: {Low-Latency} video processing using thousands of tiny threads," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 363-376. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi
- R. Graves, T. H. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner, D. Okaya, P. Small, and K. Vahi, "Cybershake: A physics-based seismic hazard model for southern california," *Pure and Applied Geophysics*, vol. 168, no. 3, pp. 367-381, 2011. [Online]. Available: https://doi.org/10.1007/s00024-010-0161-6
- J. C. Jacob, D. S. Katz, G. B. Berriman, J. C. Good, A. Laity, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, T. Prince, and R. Williams, "Montage: A grid portal and. software toolkit for science-grade astronomical image mosaicking," *International Journal of Computational Science and Engineering*, vol. 4, no. 2, pp. 73-87, 2009. [Online]. Available: https://doi.org/10.1504/IJCSE.2009.026999
- Apache spark. [Online]. Available: https://spark.apache.org/
- Apache hadoop. [Online]. Available: https://hadoop.apache.org/
- Htcondor. [Online]. Available: https://htcondor.org/

References (cont.)

- Apache airflow. [Online]. Available: https://airflow.apache.org/
- Prefect. [Online]. Available: https://www.prefect.io/
- Dagster. [Online]. Available: https://dagster.io/
- Argo workflows. [Online]. Available: https://argoproj.github.io/workflows/
- Aws step functions. [Online]. Available: https://aws.amazon.com/en/step-functions/
- Azure durable functions. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview
- Google cloud workflows. [Online]. Available: https://cloud.google.com/workflows
- Q. Jiang, Y. C. Lee, and A. Y. Zomaya, "Serverless execution of scientific workflows," in *International Conference on Service-Oriented Computing*. Springer, 2017, pp. 706-721. [Online]. Available: https://doi.org/10.1007/978-3-319-69035-3_51
- E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," in *Proceedings of the 2017 symposium on cloud computing*, 2017, pp. 445-451. [Online]. Available: https://arxiv.org/abs/1702.04024
- D. H. Liu, A. Levy, S. Noghabi, and S. Burckhardt, "Doing more with less: Orchestrating serverless applications without an orchestrator," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 1505-1519. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/liu-david
- Docker. [Online]. Available: https://www.docker.com/
- Redis. [Online]. Available: https://redis.io/
- Docker resource constraints. [Online]. Available: https://docs.docker.com/engine/containers/resource_constraints/