ArmoniK : An Open-Source Solution for Computation Orchestration and Distribution

Jérôme Gurhem - Wilfried Kirschenmann
Aneo, Boulogne-Billancourt, France

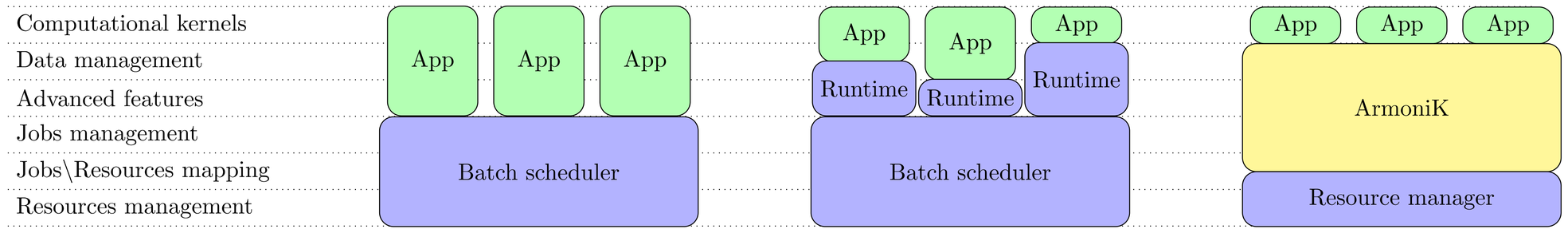
Objectives

- Provide an Open-Source, scalable platform for executing distributed workloads efficiently on heterogeneous infrastructures
- Simplify the development and deployment of distributed computing codes
- Maximize resource utilization across private/public clouds and HPC clusters
- Provide a high-level abstraction for developers

Key Benefits and Outcomes

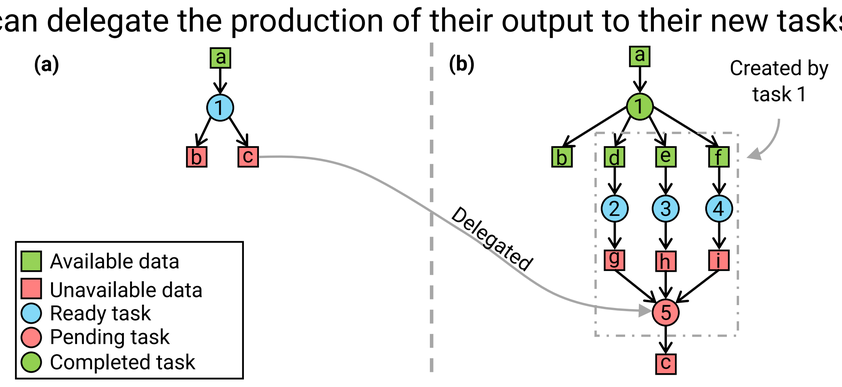
- Smart orchestration ensures efficient task execution at scale
- Modular, elastic architecture adapts to workload variation
- Built-in observability enables reliability and performance monitoring
- Empowers the next generation of high-performance, data-intensive applications

ArmoniK Positionning in HPC



Dynamic Graph

- Dependency graph is not fully known when scheduling starts
- Submissions can happen anytime
- Tasks can submit new tasks
- Tasks can delegate the production of their output to their new tasks

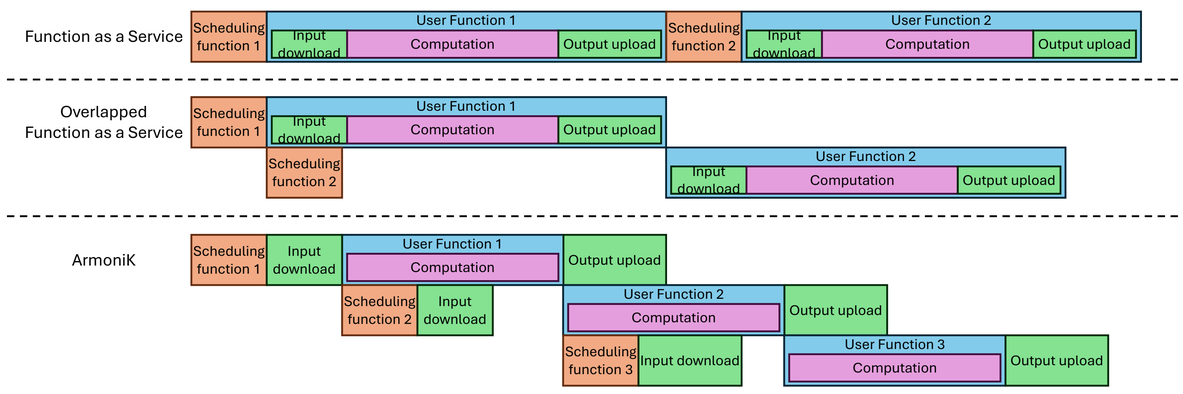


Other features

- **Observability**: GUIs, CLIs, monitoring APIs, metrics, logs, and traces to understand of the state of the system
- **Portability**: Easy to transfer an application from one environment to another
  - Officially supported languages: C#, C++, Python, Rust, Java, and JavaScript
  - Tasks on different architectures (x86, ARM, GPU, Linux, Windows), applications, environments
- **Malleability**: Support dynamic reconfiguration of the number of allocated resources during execution without interruption
- **Resource Sharing**: Allow sharing resources between applications to execute as many as possible at the same
- **Modularity**: Modules can be swapped without modifying ArmoniK's code to suit user neeeds and constraints

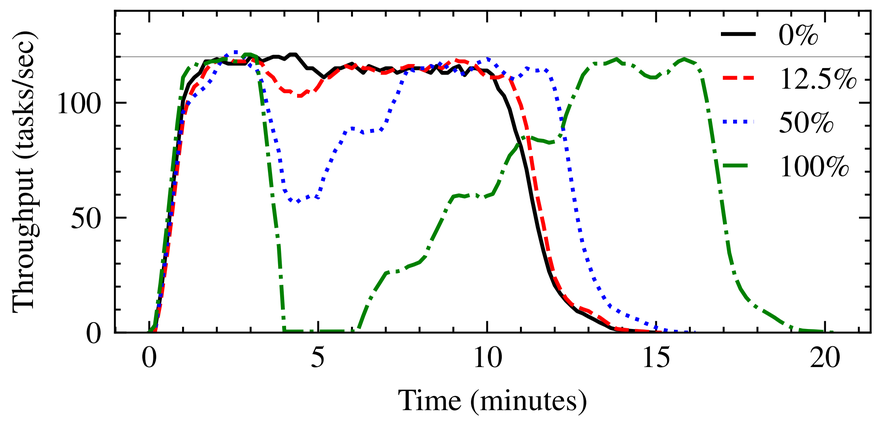
Computations/Comm Overlapping

- ArmoniK is responsible for tasks input and output data management
- Allow for automatic communication + scheduling/task execution overlapping
- Automatic Uncoordinated Checkpointing



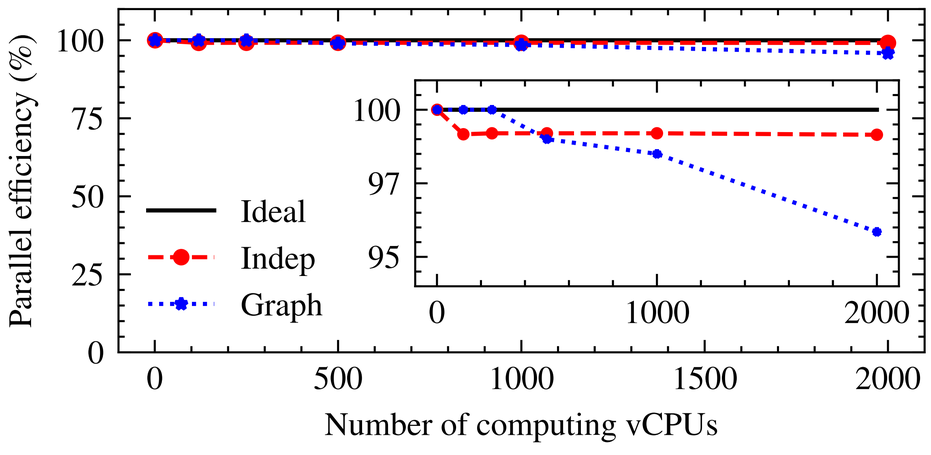
Fault Tolerance

- Works without interruption even when one or more nodes fail
- Allow support for preemptible computing resources
- Automatic and efficient task retry on failure
- Each curve represents a percentage of preempted instances



Throughput Scalability

- Indep : independent tasks workload
- Graph : nested fork-join workload



Low Round-trip Latency

- Cumulative distribution functions (CDFs) of round-trip latency
- Batched submissions of 1, 10, and 100 independent zero-work tasks