# Analysis of Distributed Execution of Workloads

The motivation for writing this paper is to compare the architectures and capabilities of resource selection and task placement at distributed systems. The author compares the AIMES middleware, Swift workflow, and integration of these two systems by measuring the time of completion of workloads that executed at diverse scale and on multiple resources. There must be a mechanism that efficiently selects and manages resources which related to the concerns such as ways to particular binding, scheduling, and distribution of tasks to the resources. These issues become more complex when time-dependent availability, heterogeneous resources, and workloads are added to the systems.

The Author compares AIMES and Swift execution strategies with metrics like time-to-completion (TTC) of a workload. The AIMES uses Execution Manager to support the development of alternative decision strategies and optimization metrics that acts with a Pilot Manager and enables late binding of both pilots and resources. The requirements of the workload's tasks are evaluated before any pilot is described and assigned to the resource. AIMES can handle multiple pilots on multiple resources with global scheduling algorithms. The Swift benefits from the app tasks strategy which defines as tasks in workflow models are independent applications and communicate with each other by data flows. The Swift programs converted to scripts and the executed with the workflow engine. The 'Coasters' system assigns computing resources to nodes and importing and exporting files from the local file system of nodes.  In this mechanism, the number of submitted tasks can be controlled by a user or with dynamic selection algorithms.

AIMES and Swift are end-to-end systems, even though, have not the same functionality in areas like workload specification, resource management, and task execution. Both of them aid users to determine multi-task workloads and then perform them on various resources. Initially, the AIMES have not a workload description language for workload specification. On the other hand, The Swift has a workflow that helps to identify tasks as functions. Moreover, The AIMES get its required informations such as historical and real-time assessments from Bundles. Conversely, Swift uses a configuration file which contains parameters that determine the parallelism. Two systems implementation is different on assigning pilots to resources. In addition, while Swift with site selection algorithm binds all tasks to resources, AIMES enable late binding tasks only to pilots (not to resources) as well as early binding of pilots to the site. Finally, the AIMES task execution strategy needs to know about the whole tasks of workload and could not handle newly added tasks to the system. On the other hand, the swift structure separates task execution and specification.

The integrated AIMES and Swift benefits from these architectural and functional similarities and differences. With the integrated system, an HTTP-based RESTful API translates Swift task descriptions to AIMES task descriptions, which are used by AIMES Execution Manager. The Swift has no information about workflow inputs, The AIMES, though, needs overall data about workloads before starting. The restful interface enables Swift's AIMES provider to submit tasks for execution when they are delivered by the Swift Workflow Engine. When the number of submitted tasks reaches a certain rate, AIMES begin to execute tasks.

The experiments conducted to study of the performance of four execution strategies by following goals: the effect of them on TTC; to discover how architectural design properties corporate with execution strategies; and to show how the integration of both systems supports workflows on distributed and diverse resources. At performing experiments with Swift and AIMES separately, (Experiment 1) with minimizing the wall time requested for each pilot and maximizes the number of pilots, we can see TTC depends on the availability of resources and the overheads introduced by managing pilots. Furthermore, the execution of the smaller bag of tasks, has a better performance. By minimizing the number of pilots and increasing the wall time, (Experiment 2) results illustrate the average performance is better. In the experiments with late binding of tasks to pilots instead of resources, (Experiment 3) experiment with one pilot for each resource endorses the decline of average TTC and an increased efficiency of the execution strategy that used for this experiment compared to Experiment 1 and 2. Experiment 4 indicated increasing the number of resources, improves the performance. The author delineates the experiments on the integration of the two systems. The analysis shows TTC increases due to the size of files that imported and exported by workflow, the executing time for workload will be higher than previous experiments. Also, by binding to active pilots and a reduction in the typical time for the activation of the first pilot, we have small error bars. By introducing data staging, due to the result of how late binding to pilots is implemented, the execution strategy of the integrated experiments performed better than the Experiment 4.

This paper has revealed several ways for future research. The concept of execution strategy needs to be generalized and its use extended to other systems. Swift's and AIMES' state models cannot be completely matched because of restrictions on sharing information. The essential prerequisite for developing integration between AIMES and Swift is extending the RESTful API to exchange information between them.