A Dynamic Event Driven Method for Customizing Scientific Workflow

Authors Name/s per 1st Affiliation (Author)
line 1 (of Affiliation): dept. name of organization
line 2: name of organization, acronyms acceptable
line 3: City, Country
line 4: Email: name@xyz.com

Authors Name/s per 2nd Affiliation (Author)
line 1 (of Affiliation): dept. name of organization
line 2: name of organization, acronyms acceptable
line 3: City, Country
line 4: Email: name@xyz.com

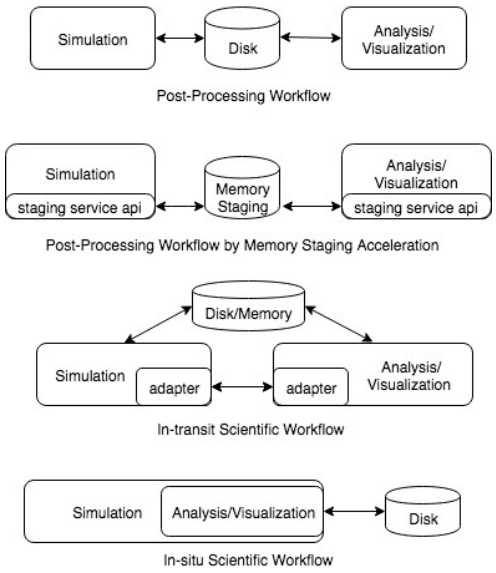

Figure 1. Scientific Workflow Pattern

***Abstract*—The abstract goes here. DO NOT USE SPECIAL CHARACTERS, SYMBOLS, OR MATH IN YOUR TITLE OR ABSTRACT.**

***Keywords*-component; formatting; style; styling;**

I. Introduction

The typical running pattern for scientific application referring to simulation, analyzing and visualization tasks are divided into post-processing and in-situ[**?**]. Post-processing use disk or memory staging service to transfer the intermediate data. In-situ processing is a different approach to solve the gap between the computation and communication performance of HPC infrastructure without involving disk storage[**?**]. Several works are focusing on the in-situ pattern and how to leverage the scientific application[**?**], [**?**] by the in-situ pattern. Figure 1 shows the typical scientific workflow pattern.

The main task of scientific workflow aims to solve the problems of how and when to run the task in an automatic way as far as possible. Post-processing can be supported well by workflow based DAG (Directed Acyclic Graph) with fixed dependency before workflow run. Even though there are all kinds of work focusing on the intermediate component such as data staging service or in-situ adaptor to leverage specific scenarios (we will discuss this in Background and related work) there are few works explore how to support hybrid pattern in portable, flexible and scalable way(the aim of the experiments). For example, we could use the in-situ part in the simulation to identify if some interesting things happen, if the interesting happens, the in-transit pattern can be used to notify analysis, analysis and visualization component to load data from staging service can be triggered at the same time.

One challenge for scientific workflow is how to support the complex integration of application patterns including in-situ, in-transit and post-processing tasks[**?**]. There are two questions needed to be solved in this scenario. (1) Tasks in a workflow is not in same abstraction level, for example, the in-situ part can be a thread in a program and the post-processing part is another program (2) When to run task is unknown before workflow start, the task could be triggered several times or even not be triggered based on the content of intermediate data application.

Our work provides a method based on event programming to leverage the hybrid scientific pattern including in-situ and in-transit pattern, we also implement an event driving workflow tool to compose the scientific application and evaluate its effectiveness in experiment part. The main advantages of this event-driven workflow include:

(1)The dependency could be defined in a more flexible form based on execution sequence or content of intermediate data.

(2)The workflow tool could help the user to express the dependency and control for tasks with different abstraction level and running pattern such as in-situ, in-transit and post-processing.

(3)There is also potential to combine this workflow with low-level resource scheduler system to leverage whole running task of the scientific workflow (more detailed after experiments finishing, maybe provide some good practice to implement the hybrid workflow including in-situ in-transit and post-processing)

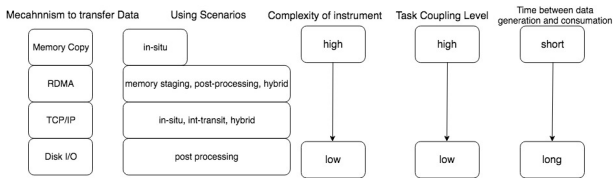

Figure 2. Classification for scientific application running pattern

The rest paper is organized as follows, Section II introduce the background and the related work of event-driven workflow, Section III introduce the design and implementation of workflow framework. Section IV presents the experiments to show the performance and effectiveness of the framework. Section V introduce the conclusion and the future work of the work.

II. Background and Related Work

In this section, we briefly introduce the relevant work of scientific application from the aspect of task dependencies, then we introduce the workflow system that supports those using scenarios and explain the shortage of current workflow system, at last, we will introduce the pub-sub mechanism and how it will be used in the workflow system.

A. Types of Scientific Application

The typical running pattern for scientific application referring to simulation, analyzing and visualization tasks are divided into post-processing and in-situ[**?**]. Post-processing use disk or memory staging service to transfer the intermediate data. In-situ processing is a different approach to solve the gap between the computation and communication performance of HPC infrastructure without involving disk storage[**?**]. Several works are focusing on the in-situ pattern and how to leverage the scientific application[**?**], [**?**] by the in-situ pattern. We clarify properties of the scientific application running pattern by several essential underlying perspectives which are shown in Figure 2.

Every column in Figure 1 represents a specific dimension of scientific application pattern. For traditional disk I/O based post-processing, the visualization/simulation will load data outputted by previous dependent tasks from disk, the complexity to modify the source code is low because it is straightforward to output data into disk files. There is also comparatively low coupling level [**?**] because the task to output data and the task to load the data can be put into the different programs. Because the data will be consumed after the simulation finish, the time between data generation and consumption is also long compared with other using scenario. Besides, the post-processing can be leveraged by using RDMA and staging service to decrease I/O overhead.

If several tasks are included in one program, the data can be transferred by the network using MPI or other dedicated network libraries (in-transit/in-situ workflow), coupling level will increase by this way and instrument code to transfer data is non-trivial. The specialized adaptor such as the Visit Libsm [**?**] and Paraview Catalyst [**?**] can be used synchronously in simulation code to transfer data to fit in the visualization component and the time between data generation and data consumption will decrease by avoiding disk I/ O. Besides, the in-transit pattern can be leveraged further by in-situ way if visualization and simulation are composed together using memory copy to transfer data.

There is no absolute preferred pattern for any using context and there is some tradeoff for every pattern according to Figure 1. In-situ and in-transit analysis do not displace the need for post-processing pattern even if there is large I/O cost for scientific workflow[**?**]. For example, post-processing is better when scientists need to use visualization tool to observe the simulation data from different perspective manually to get more insights. For the in-situ and in-transit application, the disadvantage is that the performance effect for the simulation and not all coupling task is suitable for the in-situ pattern. The Hybrid workflow including several different types of running pattern in Figure 2 is more wildly used in the real scientific workflow which is composed of different level of coupling tasks in order to leverage the advantage of the specific running pattern and minimize their disadvantages.

B. Types of Scientific Workflow Pattern

Considering the scientific application in [**?**], the execution of tasks in workflow could be represented as DAG, every node represent the execution of the task and every edge represent the sequence of the execution between tasks. For workflow tool such as [**?**], [**?**], the abstraction of DAG should be fixed before the application running, the user should grasp the topology of the task dependency in advance and execute the workflow by submitting the description file, then workflow will schedule and execute the task according to this DAG blueprint. The intermediate file was used to control the execution of the typical dependency pattern[**?**].

There are other types applications without fixed and explicit task dependency pattern before the task running, the task may or may not be triggered during task running and it depends on the content of the data instead of the input/output data files in those cases. For example, in simulation-analyzing-visualization workflow, a large amount of simulation data will be generated in short time, some simulation will also running several days, there are some redundant data in the output of the data but only some of them are useful data. If we only want to visualize the data with the average value larger than the specific threshold, the picture will be plotted only when data satisfy our predefined condition. There are several strategies for workflow system to address this issue, one is post-processing which will load all the simulation data into the disk and then process the data in separate tasks. But this solution only useful for

small-scale simulation, the increasing data, size and limited storage and bandwidth make high fidelity post-processing impractical[**?**]. In situ workflow is the latest trend to solve this issue[**?**]. There are several intermediate components to leverage the in-situ workflow[**?**], [**?**] there are works exploring how to decompose original algorithm into in-situ method such as[**?**] some works are exploring the properties of workflow at exascale[**?**] but few works focusing on how workflow support different type of in-situ tasks involving flexible dependencies relationship.

C. Types of Dependencies between Workflow Tasks

One critical distinction between the traditional workflow(without in-situ task) and the in-situ workflow is the construction of task dependencies. For traditional workflow such as [**?**], the dependencies are determined by input and output files between tasks, for example, if the output file of task A is the input of task B there is a dependency between task A and task B namely task B needed to be started after taskA finish. In the in-situ/in-transit/Hybrid workflow, the dependency will be described in much finer granularity and more flexible way. For example, task B will start after one loop iteration of taskA finish, or taskB needed to be started when the output of taskA satisfy the specific condition. According to the Figure 1 in the introduction, this is a kind of hybrid pattern: the in-situ part will be used to detect if the interested happens and then the in-transit pattern will be triggered to analyze the data. Dedicated communication libraries needed to be used to send the message between different tasks to synchronize the execution of the task, namely to decide when the task is supposed to be started. No matter for the traditional task or in situ tasks, the task dependency pattern could be divided into three main types [**?**], [**?**] which are shown in Figure 2.

For traditional workflow, the intermediate data are files on the disk and the work queue[**?**] could be used to control the execution of tasks, for in-situ workflow, one solution is to view the intermediate data as series of event messages which will be used to control the task execution sequence generated by in-situ tasks or infrastructure running the tasks.

D. Pub-Sub paradigm and event-driven workflow pattern

Once we assume event message as a special intermediate data used to synchronize the task execution, Pub-Sub paradigm [**?**] could satisfy the requirements to run fine granularity task dependency by messaging communication in the distributed way, There are also all kinds of implementation in research and industry using event message broker based on those pub-sub-notify architecture[**?**], [**?**], [**?**], but there are few scientific workflow systems combining the task dependency and event driving programming together. Some works such as [**?**] use this mechanism to leverage the in situ workflow, but they only support the specific underlying library and the types of task dependency pattern is also limited. A canonical pub-sub mechanism works in three steps , assume there are two component namely subscriber S and publisher P (1) S will subscribe one or more interesting events into the pub-sub broker (2) P will push events into the pub-sub broker (3) pub-sub broker will notify S if pushed event match the subscribed event. There exists a dependency between publisher and subscriber in this paradigm. If we use this programming model to compose workflow, the task could be a publisher or subscriber at in the workflow, there will be less restriction on task because any type of tasks could generate the event used to construct the dependency , the task can be in-situ part of the task or jobs running by sbatch system flexibly according to different using scenarios. By this task composing model, This kind of event-driven workflow should support the task paradigm in Figure 3. One advantage to use event message to express dependency is the flexibility for dependency granularity, the event message is an abstraction of the user-defined dependent data bock which could be the disk files or data instance in memory. There are following scenarios to use event programming to leverage scientific workflow:

(1)Event messages could also represent the properties of data and runtime, for example, an event could represent data in a specific domain is larger than a threshold

(2)The status of underlying infrastructure, the start, stop of runtime system(batch runtime system or container runtime system)

(3)In-situ, in-transit and post-processing tasks can be composed together in a workflow by event message abstraction

The broadcaster and pipeline pattern are supported by pub-sub-notify mechanism naively but there is no support for aggregation pattern. For typical pub-sub mechanism, if subscriber subscribe multiple events such as eventA, eventB eventC then any publishing of those three events will notify the subscriber and trigger execution of task, however, in workflow aggregation scenario, the specific subscribed runtime is supposed to be triggered when all eventA , eventB and eventC are satisfied and published. We will discuss the details of pub-sub event store and relevant runtime client to support all of those patterns in implementation part.

III. Implementation

We introduce the architecture of event-based workflow system and then discussed the details for every component. Figure 3 shows the architecture of event-driven workflow and it's work procedure.

The user view and the workflow frame view are shown in Figure 3, User needs to provide the initial configuration files which include the path of the job script and the event that every task interested, those events will be subscribed into the event store. The operator is the in-situ part integrated into the simulation code which is used to send subscribe and publish request during the simulation run. When specific events are published, the subscribing end will be notified and the job

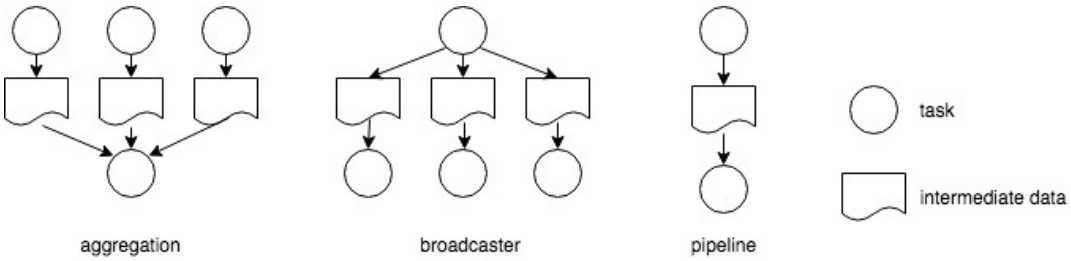

Figure 3. Typical workflow pattern

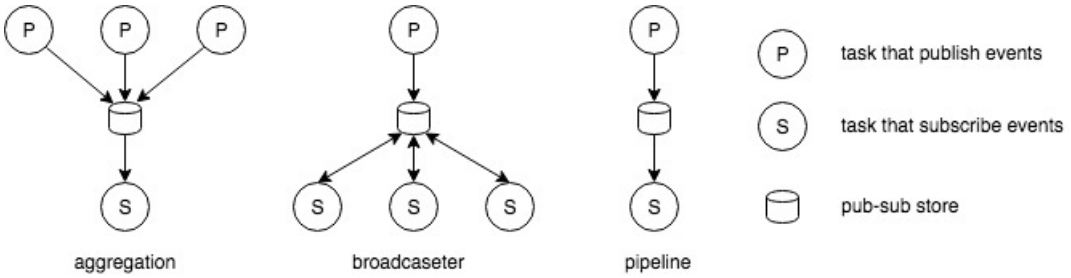

Figure 4. Typical event-driven workflow pattern

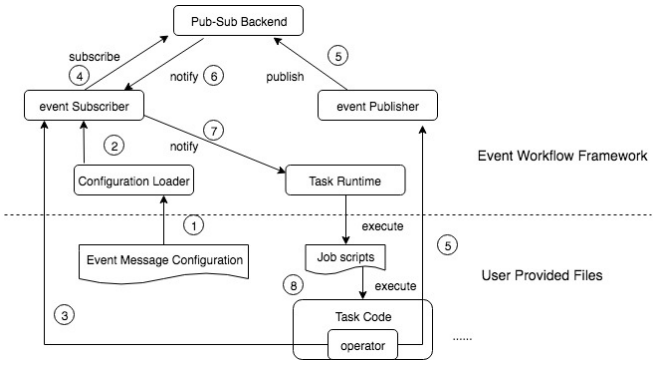

Figure 5. Event driven workflow framework archetecture

scripts will be started by task runtime. Several typical and in-situ using scenarios can be supported because this event pub-sub backend support pub-sub in 1:1, 1:n and n:1 manner and a user-defined event also provide flexibility to express the task dependency in fine-grained ways.

(1) Traditional predefined static dependency tasks can be supported by running operator before task start running (subscribe event) after the task finish (publish event), the task will start running only when the specific event happens.

(2) The granularity of data dependency can be defined by the user in a flexible way to avoid post-processing and decrease all the workflow running time. For example, when every thread generates specific domain data, the visualization task can be notified and triggered immediately when all the subscribed events are satisfied.

More detailed using scenarios will be discussed in experiments and evaluation. We will introduce the key component for the following parts.

A. Optimised Pub-Sub Backend

In this part, we introduce the design and implementation of optimized pub-sub mechanism and discuss how it supports both traditional and in-situ task dependency pattern in a workflow. Besides, we also leverage the canonical pub-sub mechanism and make it support the aggregation(fan-in) and broadcaster(fan-out) pattern at the same time, namely one event subscriber could subscribe multiple sub-event messages and specify the triggering conditions, when all the events are published specific times, the subscribed events will be triggered subsequently. The simplified underlying data structure is shown in Figure 6, eventually, we need to maintain several core maps to keep the mapping relation from one client id to several subscribed events and from one event to several client ID that interested to it. besides, we also need to use another map to record the published times for every event and use this info to check if the specific client should be notified. the subscribe and publish method is described as follows:

(1) event subscribe: For every subscribe request, there is a new id will be unique id will be generated and stored at back end, the event that this client interested will also be recorded and associated map will be initialised, there is a housekeeping logic in every subscribe method keeps checking the satisfactory flag, request will block here if the flag is not satisfied, otherwise the notify status will be returned back to client.

(2) event publish: When there is publish request, the

backend will parse the event message in a request and retrieve the event from the map of an event to subscribe client id. For example, if we assume the publishing event is EVENTA. For every client id, we could get the event set it subscribed(every client is associated with an event message set), we could find EVENTA from this set and increase one for the number it is pushed, then we could check the required triggering number in published event, if the number equals to the required number, we could set the satisfied flag as true.

B. Event Configuration and Data Operator

C. Task Runtime

Task Runtime is designed to support multiple drivers to execute a task. The input of the task runtime is the driver name and the path of the running scripts. If the driver is the container solution such as Singularity or Docker, the execution path could be the name of the image.

IV. Evaluation and Experiments

We have evaluated the performance of the event driven workflow tool on Caliburan cluster at Rutgers Discovery Informatics Institute. The Caliburan cluster is based on a FatTwin SuperServer system. It has 560 nodes, each with two Intel Xeon E5-2695 v4 (Broadwell) processors, 256 gigabytes (GB) of RAM, and a 400 GB Intel NVMe drive. Overall, the system has 20,160 cores, 140 TB of memory and 218 TB of non-volatile memory. The performance is 603 TFLOPS with a peak performance of 677 TFLOPS[**?**].

A. Performance Evaluation

The first experiment is measure the latency of task triggering time. The test using synthetic workflow include two tasks, taskB depends on taskA. Our test based on the workflow pattern in Figure 2 which include aggregation, broad caster and chained pattern. For every latency, we also compare our workflow tool with the other workflow tool Swift[**?**] and Makeflow[**?**].

For chained pattern when taskA.X finish taskB.X will be triggered, we increase the number of coupling pairs of (taskA.X, taskB.X) to test the latency.

For aggregation pattern, there are several taskA.X and one taskB, when all taskA.X finished, task.B will be triggered and we will test waiting time of the taskB with the increasing number of taskA.X

For broadcaster pattern, there are one taskB and several taskA.X, when taskA finish, all the task.B will be triggered and we will test the average waiting time of the taskB with the increasing of the taskB.X.

B. hybrid workflow using scenarios

In this part, we use a real use case to show how event workflow tool compose different abstraction of task and

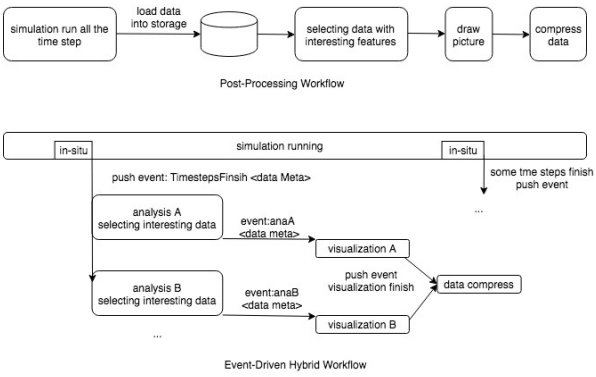

Figure 6. Scientific Workflow Pattern

leverage the scientific application. Figure ? shows the different between the traditional post-processing workflow and the hybrid event driven workflow. We could notice that in-situ task and the post-processing task could be composed together by event driven workflow and the synchronization between different task is also minimized by event messages.

V. Conclusion

support more types of task running tool

Future work: the language/compiler tool to facilitate the creation of the event configuration

make pub/sub in more scalable level and let it more distributed

challenge, traditional dependency is in layered way but new pattern is in flattern way

more suitable for the task with run in several time and specific things will happen for several times. not cascade pattern

integrate the event driven workflow with in industry in-situ visualisation solution such as libsm in visit and caalyst in paraview