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

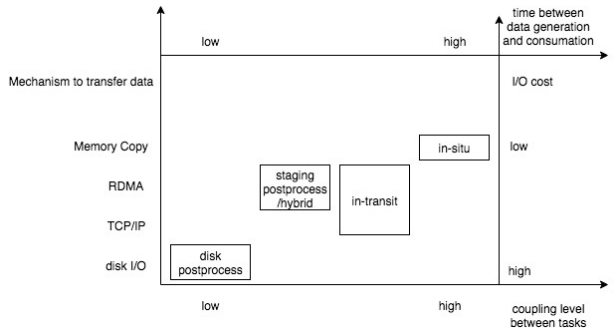

Figure 1. classification for scientific application running pattern

***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

In-situ processing is a solution to solve the gap between the computation and communication performance of HPC infrastructure for several decades[8]. Several works are focusing on the in-situ solution and how to make it leverage the scientific application[5], [15]. Typical subtypes for simulation-analyzing-visualization application[8]: post-processing , co-processing, concurrent processing, Hybrid. There are other dimension to divide the application such as if they are running within same program [11]. Based on those definitions of the concept of in-situ, in-transit, hybrid scientific application, we clarify properties of every solution by several essential underlying perspective which are shown in Figure 1, to be more specific:

The post-processing analysis with disk using is the naive and traditional way to compose a scientific workflow. The visualization/simulation will load the disk data outputted by previous dependent tasks. There is a minimum degree of coupling (largest independency) between two tasks in this context because the task of data produces don't need any information about the task which consuming data.

The post-processing analysis with staging server works by constructing pipeline based on data management service. The I/O cost can be decreased by leverage the function of staging service, but it required some modification of simulation in order to using API provided by staging service. Further optimization could be adopted in staging service to decrease I/O cost relatively such as data locality and placement strategies.

If tasks are composed within one program, the data could also be transferred by the network such as using MPI or other dedicated network library (in-transit workflow) this will increase the coupling level and instrument simulation code in much higher level. The data could also be transformed by the specialized adaptor to transform the data structure.

Within one program, the data could also be transferred by much higher level coupling namely run visualization code synchronously after each iteration which increases coupling level and data transfer speed (in same memory space).

There is no absolute preference solution for any using context and every solution has tradeoff in Figure 1, in situ and in transit analysis does not displace the need for post-post processing pattern even if there is large I/O cost for scientific workflow[5]. For in-situ and in-transit application the disadvantage is that the in-situ code will influence the performance of the simulation, not all coupling task is suitable for the in-situ pattern. The Hybrid workflow is more useful in real scientific workflow which is composed of different level of coupling task inorder to leverage the advantage for specific task and minimize their disadvantages. One challenge is how the workflow to support the complex integration of application patterns in Figure 1. 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).

Our work provides a method based on event programming to leverage the hybrid scientific pattern including in-situ and in-transit part, we also implement an event driving workflow tool to compose the scientific application and evaluate its effectiveness in Experiment part. We also show the 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)

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 it's using in the workflow system.

A. Types of Scientific Application and Workflow System

Considering the scientific application in [13], the execution of tasks in workflow could be represented as DAG (Directed Acyclic Graph), every node represent the execution of the task and every edge represent the sequence of the execution between tasks. For workflow tool such as [13], [16], 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[7].

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[4]. In situ workflow is the latest trend to solve this issue[10]. There are several intermediate components to leverage the in-situ workflow[9], [4] there are works exploring how to decompose original algorithm into in-situ method such as[6] some works are exploring the properties of workflow at exascale[10] but few works focusing on how workflow support different type of in-situ tasks involving flexible dependencies relationship.

B. 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 [3], 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 [3], [7] which are shown in Figure 2.

For traditional workflow, the intermediate data are files on the disk and the work queue[3] 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.

C. 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 [12] 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[1], [14], [2], but there are few scientific workflow systems combining the task dependency and event driving programming together. Some works such as [14] 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

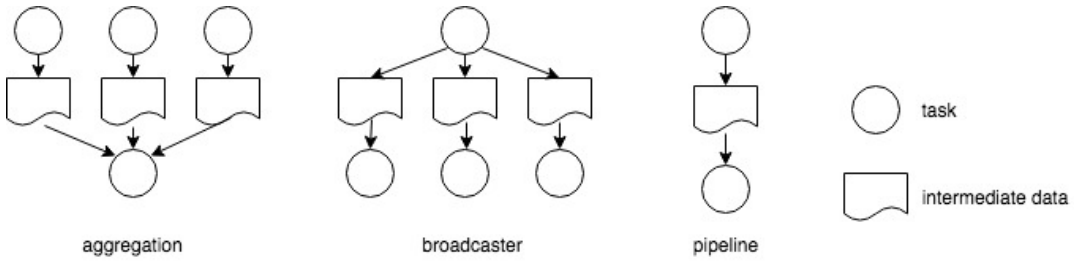

Figure 2. typical workflow pattern

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:

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, is subscriber subscribe multiple events such as eventA, eventB eventC, and 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 published and satisfied. 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 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 house-keeping 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

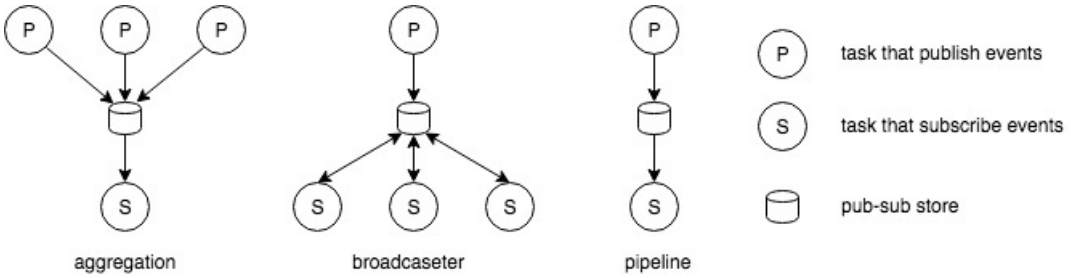

Figure 3. typical event-driven workflow pattern

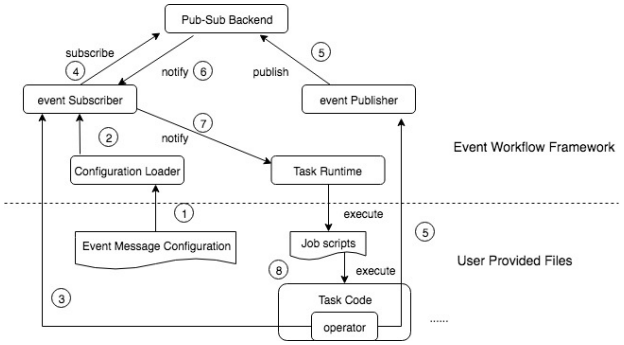

Figure 4. event driven workflow framework archetecture

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. 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

References

[1] A unified data-driven approach for programming in situ analysis and visualization. https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe. 2015.

[2] redispubsub. https://redis.io/topics/pubsub, 2017.

[3] Michael Albrecht, Patrick Donnelly, Peter Bui, and Douglas Thain. Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, page 1. ACM, 2012.

[4] Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. Paraview catalyst: Enabling in situ data analysis and visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pages 25–29. ACM, 2015.

[5] Andrew C Bauer, Hasan Abbasi, James Ahrens, Hank Childs, Berk Geveci, Scott Klasky, Kenneth Moreland, Patrick O'Leary, Venkatram Vishwanath, Brad Whitlock, et al. In situ methods, infrastructures, and applications on high performance computing platforms. In *Computer Graphics Forum*, volume 35, pages 577–597. Wiley Online Library, 2016.

[6] Janine C Bennett, Hasan Abbasi, Peer-Timo Bremer, Ray Grout, Attila Gyulassy, Tong Jin, Scott Klasky, Hemanth Kolla, Manish Parashar, Valerio Pascucci, et al. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 49. IEEE Computer Society Press, 2012.

[7] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*, pages 1–10. IEEE, 2008.

[8] Hank Childs, Kwan-Liu Ma, Hongfeng Yu, Brad Whitlock, Jeremy Meredith, Jean Favre, Scott Klasky, Norbert Podhorszki, Karsten Schwan, Matthew Wolf, et al. In situ processing. 2012.

[9] Ciprian Docan, Manish Parashar, and Scott Klasky. Dataspaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing*, 15(2):163–181, 2012.

[10] Matthieu Dreher, Swann Perarnau, Tom Peterka, Kamil Iskra, and Pete Beckman. In situ workflows at exascale: System software to the rescue. In *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization*, pages 22–26. ACM, 2017.

[11] Matthieu Dreher and Tom Peterka. Decaf: Decoupled dataflows for in situ high-performance workflows. Technical report, Argonne National Lab.(ANL), Argonne, IL (United States), 2017.

[12] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2):114–131, 2003.

[13] Nick Hazekamp and Douglas Thain. Makeflow examples repository. https://github.com/cooperative-computing-lab/makeflow-examples, 2017.

[14] Tong Jin, Fan Zhang, Manish Parashar, Scott Klasky, Norbert Podhorszki, and Hasan Abbasi. A scalable messaging system for accelerating discovery from large scale scientific simulations. In *High Performance Computing (HiPC), 2012 19th International Conference on*, pages 1–10. IEEE, 2012.

[15] Ron A Oldfield, Kenneth Moreland, Nathan Fabian, and David Rogers. Evaluation of methods to integrate analysis into a large-scale shock shock physics code. In *Proceedings of the 28th ACM international conference on Supercomputing*, pages 83–92. ACM, 2014.

[16] Michael Wilde, Mihael Hategan, Justin M Wozniak, Ben Clifford, Daniel S Katz, and Ian Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652, 2011.