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

***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

[scientific applications and it's workflow]

Workflow is typically defined as a sequence of operations or tasks needed to manage a buiiness or computational activity, the scientific workflow tools aims to support the execution of scientific applications mainly including tasks such as scientific simulations, analysis and visualization etc. one character is the large data exchange between those applications. In-situ workflow could be composed by exchanging those intermediate data by memory/storage hierarchy and network of a high-performance computing (HPC) [6]. Those intermediate data can be fully used to guide the excution of the scientific workflow.

[general description about the question we want to solve]

According to the communication model between tasks, the scientific workflow could be divided into integrated workflow and connected workflow.[7]. Integrated workflow run all tasks in one MPI program and Connected workflow coordinate tasks separated into different programs not sharing common communicator. One advantage for connected workflow is customizing the occasion to start the tasks based on the influential events in workflow.

Typical workflow engine follows predefined DAG (Directed Acyclic Graph) to start every tasks. Every node in DAG represent a task and every edge represent the dependency between task. According to the method to establish the dependency, the workflow could be divided into control-driven workflows and data-driven workflows[17]. Data driven is constructed based on the data dependency between two tasks, namely the output of first task is the input for second task. Compare with the control-driven flow (Task A must run after Task B), the data driven flow provide a view to describe the task dependency in much finer granularity way.

[challenge of scientific workflow for data driven dependency]

One scenario in complex scientific application is that the dependency is constructed according to the status of task itself and the contents of intermediated data. For example, when the task fail or intermediate data satisfy some conditions, extra tasks will be triggered.

Some project have already provide the solution for those dependency construction such as Decaf [8] and Argo[15] but they mechanism of how to construct and control fine granularity depedency is not fully explored. We leverage the dynamicity of the connected workflow based on the event driven pub-sub architecture and let user describe and subscribe their interested events for every task. When those event based on data content happens in workflow, the related task will be triggered based on those events.

slurm is the common tool For HPC users, even if it provide the strigger function to assign a trigger command when some events happens[1], but only the user with root permission could use the slurm trigger function and the types of the event is also limited and could not be customized.

advantages to use the container Singularity(1 convenience for development and production 2 provide more dynamic way to monitor the runtime of the process and publish specific status by event message)

[our solution and main contribution]

There are several for communication between pub-sub system based on space, time synchronization[9]. We leverage these advantages by using it to express data dependency between workflow tasks. In this paper, we mainly make the following contributions:(planned)

1.Provide the expression model to let user describe the event and the logic to judge how this event will happen.

2.Implement a event driven pub-sub system which could start the task at the occasion of the subscribed event happen in system

3.Provide the mechanism to acquire the events from the running tasks status and the intermediate data of the workflow.

4.Validate the effectiveness of workflow tools in different use scenarios.

dynamic register the event in running time is also another contribution (the function of unsubscribe could also be added into the system)

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

II. Background

In this section, we briefly introduce the types of scientific application from aspect of task dependencies, then we introduce the workflow system that support those using scenarios and explain the shortage of current workflow system.

A. Types of Scientific Application and Workflow System

Considering the scientific application in [11], 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 in [11], the abstraction of DAG could be acquired before the application running, then workflow will schedule and execute the task according to this DAG blueprint. There are some workflow framework such as [**?**], [19] was designed to support these kind of dependencies fixed application. The intermediate file was used to control the execution of the typical dependency pattern[5].

There are other types applications without fixed and explicit task dependency pattern before the task running, the task triggering depends on the content of the data instead of the input/output data files in those cases. For example, in simulation-analyzing-visualization workflow, 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 visualise the data with average value larger than specific threshhold, 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 disk and then processing 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[7]. There are several intermediate component 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[7] 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 distinguish 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 is determined by input and output files between tasks, for example, if the output file of task A is the input of the task B there is dependency between taskA and taskB namely taskB needed to be started after taskA finish. In situ workflow, the dependency will be in fine granularity and 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 specific condition. Dedicated communication libraries needed to be used to send message between different tasks to synchronise 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 [**?**], [5] which are shown in Figure 1.

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, the intermediate data is event message generated by in-situ tasks and the task dispatching mechanism will be discussed in implementation.

C. Pub-Sub paradigm and event driven workflow pattern

If we treat the event message as a special intermediate data used to synchronise the task execution, Pub-Sub paradigm [9] could satisfy the requirements to run fine granularity task dependency by messaging communication in distributed way, but there are few scientific workflow system combining the the task dependency and event driving programming together. Some works such as [13] use this mechanism to leverage the in situ workflow, but they only support 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 interested 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 mach the subscribed event. There exists a dependency between publish 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 2:

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 event such as eventA, eventB eventC, and publishing of those three event will notify the subscriber and triggering execution of task, however, in workflow aggregation scenario, the specific subscribed

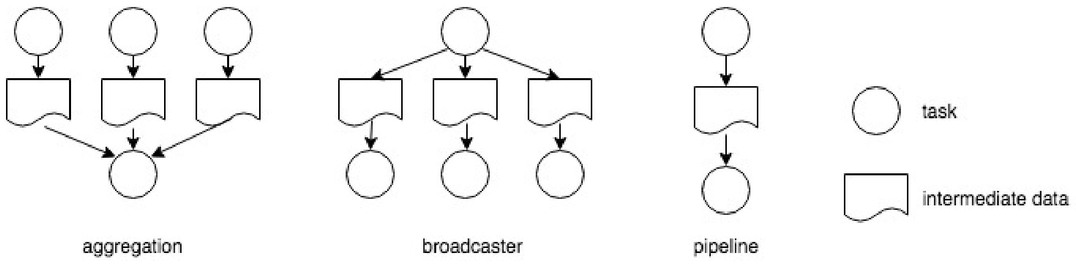

Figure 1. typical workflow pattern

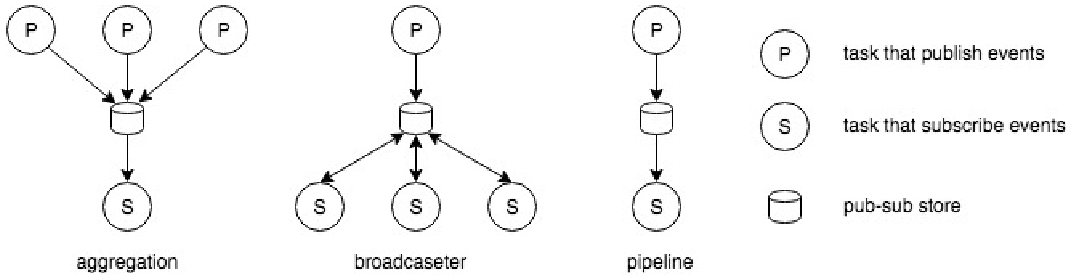

Figure 2. typical event driven workflow pattern

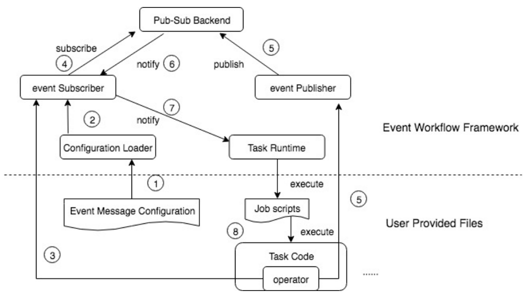

Figure 3. event workflow archetecture

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 pattern 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.

A. *Optimised Pub-Sub Backend*

In this part, we introduce the design and implementation of optimised pub-sub mechanism to support both traditional and in-situ task dependency patten in workflow.

Two key algorithms for event published subscribed. We leverage the canonical pub-sub mechanism and make it support the fan-in and fan-out pattern at the same time, namely one event message could subscribed multiple sub event message and specify the triggering conditions. When the all the events are published specific times, the subscribed events will be triggered subsequently.

Algorithm 1 Algorithm for event Subscribe

```
Input: Event List (EL), subtoClientMap(event, set(clientid)
    , clienttoSubMap(clientid, EventPublishTimeMap(event,
    publishedTimes))
Output: subscribeStatus
 1: for event in EL do
 2:   if key clientId exist in clienttoSubMap then
 3:     clienttoSubMap[clientId][event]=0
 4:   else
 5:     init EventPublishTimeMap[event]=0
 6:     clienttoSubMap[clientId]=EventPublishTimeMap
 7:   end if
 8:   if key event in subtoClientMap then
 9:     subtoClientMap[event].insert(clientId)
10:   else
11:     init set s and insert clientId
12:     subtoClientMap[event]=s
13:   end if
14: end for
15: return subscribeOk
```

B. *Data Operator*

C. *Task Runtime*

/////////////old stuff//////////////////

The design and implementation details are presented in this section.

D. Dynamic Task Manager of Event Workflow

The typical event driven mechanism works based on the observe pattern, the subject component will maintain a list of observer component, if the state of the subject component changed, the notify function will send to the associated observer components. We define and optimize our task manage based on this observe pattern, specifically, every task manager could be the subject and object component at the same time, different task could compose a workflow chain and even be organized into more complex workflow. Every task manager could be create/updated/deleted dynamically during the process of the system running. User could create/update/deleted the description file of the task manager and the instance of the task manager will be changed correspondingly.

Three abstraction: TaskManager, Communicator, Operator.

E. Event messages monitoring

Event could be divided into following parts according to the event sources: Runtime level events which created by the runtime of the tasks such as, TaskStart, TaskFail, TaskFinish. Taskelvel which is created by tasks during the task running in fine granularity such as TimeStepWrittingFinihs or DataProcessing Finish. Third party events, the events could be customized by users, when data satisfy specific logic, the event could be generated and pushed. Other platform events such as DiskFull or other events related to the running of the workflow tasks.

F. Event actions aggregation and matching

G. Event message generating

IV. Evaluation and Experiments

For traditional application, just describe the solution and evaluate the typical five pattern: latency time,

For in-situ, using an experiments and several evaluation

A. System Performance and Scalability

1) Event latency experiments. This experiments measures the scalability of (PTT) time between event published and task triggering.

we use synthetic application pairs to test this scalability.

The graph shows the results of the experiments, with the increasing of the application pairs:

the latency show what tendency?

what is the maximum number of clients/events that the system can support?

B. Effectiveness in Scientific applications

For this part we verify the effectiveness for our workflow engine in two different types of scientific applications according to the dynamics of the dependency. The dependency of the first type of application is already fixed before workflow running, our work flow engine could support the typical five task organization pattern [**?**] for this scenario. The second type is in-situ and in-transit workflow with dynamic dependency construction. The time of when the dependency is constructed depends on the properties of intermediate data during the tasks running.

V. Related Work

Pub-Sub-Notify model in HPC application: The event based architecture has advantages in decouple for the aspects of time, space and synchronization[9]. Those advantages are reflected in cloud computing area and new service type such as FaaS (Function as a Service)[10]. There are also all kinds of implementation in research and industry using event message broker based on those pub-sub-notify architecture[2], [13]. However, these works did not fully use the properties of event driven architecture in terms of dynamical event matching and using event message to connect and trigger tasks in workflow dynamically.

Workflow Execution Framework for HPC application: There are two properties for the applications in dynamic workflow: the type of data intensive and the dependency between the multi-steps in workflow. There are all kinds of workflow manage tools and programming models aiming to simplify the abstraction of workflow during programming, such as Swift[19] and Decaf [8] . For these types of systems and programming models, the developer should grasp the execution topology of the workflow in advance and excute the workflow by submitting the description file of task execution topology to the workflow engine. Even though those methods could cover lots of use contexts, but one drawback is the lack of data dynamics. One similar work Meteor[12] providing a content-based event driven decouple infrastructure based on p2p network which shows the flexibility for the dynamic programming event matching and client actions triggering.

Workflow Control Pattern: The Canonical workflow pattens are summarized in different contexts[16] , for the HPC applications, the complex workflow could be constructed on several major patterns[5]. The content-based event driven architecture could support those pattern and make some of them more efficient and dynamic in execution(supplement further)

Runtime of Workflow Execution in HPC: There are all kinds of runtime frameworks including Charm++, Legion, OCR, Uintah can help researcher to express the application into AMT(asynchronised many task) model. The comparison including the prons and cons are also discussed in quantity.[20], [21],The applications and workflow can be programmed in sequential way and executed asynchronised by the scheduler provided by those runtime system, several parallel programming model even programming languages are further explored based on those runtime tools and AMT model.[14], [4], [3]. Some works [18] are focusing on how to optimize the scheduler strategies to improve the running efficiency of in-situ workflow(supplement further),

VI. 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] Makeflow examples repository. https://slurm.schedmd.com/strigger.html.

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

[3] A unified data-driven approach for programming in situ analysis and visualization. https://extremescaleresearch.labworks.org/quad-charts. 2015.

[4] Bilge Acun, Abhishek Gupta, Nikhil Jain, Akhil Langer, Harshitha Menon, Eric Mikida, Xiang Ni, Michael Robson, Yanhua Sun, Ehsan Totoni, et al. Parallel programming with migratable objects: Charm++ in practice. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 647–658. IEEE Press, 2014.

[5] 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.

[6] Ewa Deelman, Tom Peterka, Ilkay Altintas, Christopher D Carothers, Kerstin Kleese van Dam, Kenneth Moreland, Manish Parashar, Lavanya Ramakrishnan, Michela Taufer, and Jeffrey Vetter. The future of scientific workflows. *The International Journal of High Performance Computing Applications*, page 1094342017704893, 2018.

[7] 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.

[8] 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.

[9] 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.

[10] Geoffrey C Fox, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Status of serverless computing and function-as-a-service (faas) in industry and research. *arXiv preprint arXiv:1708.08028*, 2017.

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

[12] Nanyan Jiang, Andres Quiroz, Cristina Schmidt, and Manish Parashar. Meteor: a middleware infrastructure for content-based decoupled interactions in pervasive grid environments. *Concurrency and Computation: Practice and Experience*, 20(12):1455–1484, 2008.

[13] 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.

[14] Philippe Pébaÿ, Janine C Bennett, David Hollman, Sean Treichler, Patrick S McCormick, Christine M Sweeney, Hemanth Kolla, and Alex Aiken. Towards asynchronous many-task in situ data analysis using legion. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, pages 1033–1037. IEEE, 2016.

[15] Swann Perarnau, Rajeev Thakur, Kamil Iskra, Ken Raffenetti, Franck Cappello, Rinku Gupta, Pete Beckman, Marc Snir, Henry Hoffmann, Martin Schulz, et al. Distributed monitoring and management of exascale systems in the argo project. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 173–178. Springer, 2015.

[16] Nick Russell, Arthur HM Ter Hofstede, Wil MP Van Der Aalst, and Nataliya Mulyar. Workflow control-flow patterns: A revised view. *BPM Center Report BPM-06-22, BPMcenter. org*, pages 06–22, 2006.

[17] Matthew Shields. Control-versus data-driven workflows. In *Workflows for e-Science*, pages 167–173. Springer, 2007.

[18] Qian Sun, Melissa Romanus, Tong Jin, Hongfeng Yu, Peer-Timo Bremer, Steve Petruzza, Scott Klasky, and Manish Parashar. In-staging data placement for asynchronous coupling of task-based scientific workflows. In *Extreme Scale Programming Models and Middlewar (ESPM2), International Workshop on*, pages 2–9. IEEE, 2016.

[19] 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.

[20] Jeremiah J Wilke, Matthew Tyler Bettencourt, Steven W Bova, Ken Franko, Marc Gamell, Ryan Grant, Simon David Hammond, David S Hollman, Samuel Knight, Hemanth Kolla, et al. Asynchronous many-task programming models for next generation platforms. Technical report, Sandia National Laboratories (SNL-CA), Livermore, CA (United States); Sandia National Laboratories, Albuquerque, NM, 2015.

[21] Jeremiah J Wilke, Matthew Tyler Bettencourt, Steven W Bova, Ken Franko, Marc Gamell, Ryan Grant, Simon David Hammond, David S Hollman, Samuel Knight, Hemanth Kolla, et al. Asynchronous many-task programming models for next generation platforms. Technical report, Sandia National Laboratories (SNL-CA), Livermore, CA (United States); Sandia National Laboratories, Albuquerque, NM, 2015.