Adaptive Placement of Data Analysis Tasks For Staging Based In-Situ Processing

Zhe Wang*, Pradeep Subedi‡, Matthieu Dorier†, Philip E. Davis*, Manish Parashar‡
*Rutgers University {jay.wang, philip.e.davis}@rutgers.edu
†Argonne National Laboratory mdorier@anl.gov
‡University of Utah {pradeep.subedi, manish.parashar}@utah.edu

Abstract—**In-situ processing addresses the gap between speeds of computing and I/O capabilities by processing data close to the data source, i.e., on the same system as the data source (e.g., a simulation). However, the effective implementation of in-situ processing workflows requires the optimization of several design parameters such as where on the system workflow data analysis/visualization (ana/vis) as placed and how execution as well as the interaction and data exchanges between ana/vis are coordinated. For example, in the case of hybrid in-situ processing, interacting ana/vis may be tightly or loosely coupled depending on their placement, and this can lead to very different performance and scalability. A key challenge is deciding the most appropriate ana/vis placement, which depends on dynamic applications, workflow, and system characteristics that might change at runtime. In this paper, we present a framework to support online adaptive data analysis placement during the execution of an in-situ workflow. Specifically, the paper presents a model and architecture, and explores several data analysis placement strategies. Evaluation results show that dynamically choosing appropriate data analysis placement strategies can balance the benefits and overhead of different data analysis placement patterns to reduce in-situ processing time.**

Index Terms—**in-situ, in-transit, data-driven, adaptive workflow, monitor, near-real-time decision**

I. Introduction

In-situ processing addresses the gap between computing and I/O capabilities by processing data as it is generated on the same system as the data source (e.g., a simulation). *Tightly coupled* and *loosely coupled* in-situ[1] are two fundamental models for in-situ processing. In a tightly coupled system, the simulation program is linked with an in-situ library. When the API for in-situ processing is called, the in-situ library transforms the data layout as needed and executes the in-situ tasks such as data analysis or visualization (abbreviated as ana/vis), using the same computing resources as the simulation. In contrast, the simulation program in a loosely coupled system is linked with an API for data management. Once the API is called for in-situ processing, the data generated by the simulation is transferred to other remote nodes via a high-speed network, or to other processes on the same nodes via shared memory. The memory used to store the transferred simulation data, whether local to the simulation nodes or remote, is sometimes called a *data staging area*. Analysis or visualization tools can interface with a data staging service that manages the data staging area to run in-situ ana/vis. Since the simulation data has been stored at the staging area, in-situ ana/vis can run concurrently with the simulation.

[1]The definition of concepts related to in-situ processing in this paper are consistent with those presented in [4], which standardizes the terminology for in-situ analysis and visualization systems.

The effective implementation of in-situ processing workflows requires optimizing several design parameters, such as where workflow ana/vis are placed on the system, how data is exchanged between ana/vis, and how ana/vis are executed. Hybrid in-situ systems or workflows include both tightly and loosely coupled sub-systems in one in-situ workflow. For example, lightweight data analysis can run in a tightly coupled manner, and the processed data can be sent to dedicated nodes for further processing or visualization in a loosely coupled manner. Hybrid in situ systems however bring some challenges, such as the proper placement of specific in-situ ana/vis to reduce overheads and maximize performance. Such decisions require comparing the characteristics of data ana/vis before the workflow starts. However, unexpected changes in the generated data during workflow execution may introduce uncertainty regarding, for example, the execution time of a data analysis, making it more difficult for the user to select a proper configuration a priori.

We can determine ana/vis placement for a hybrid workflow [18] if the necessary information about the data distribution and associated ana/vis is known before starting the workflow. Several prior research efforts [1], [5], [6], [22], [23] have discussed policies for making decisions about whether to use a tightly or loosely coupled approach based on prior knowledge about the workflow. When little prior knowledge is available, adaptive methods can be used to monitor key parameters and use associated policies to decide the workflow decisions, such as for example, data placement based on access pattern [20] and data analysis placement based on computing or memory resources constraints [6]. The critical issue underlying any adaptive method is the accuracy of the predicted parameters used for decision-making based on workflow runtime information. Dynamically changing simulation data (and runtime system state) can make it challenging to accurately predict key parameters such as task execution time or data transfer time. However, few existing researches discuss errors in parameter prediction and decision-making in the context of data analysis placement for in-situ processing with dynamically changing simulation data and associated processing methods.

In this work, we collect information *during workflow ex-*

ecution from the data staging service and from simulation processes. Using this information, we compare the estimated in-situ processing overhead in advance and adaptively choose the data analysis[2] placement pattern that results in the smallest execution time[3]. In order to decrease errors in decision-making and provide more accurate estimations, we propose estimation algorithm and identify relations between key parameters of the workflow, such as analysis execution time, data transfer time, and simulation computation time. Furthermore, we also make data analysis placement adjustments for edge cases, such as avoiding unnecessarily waiting for a singular process to finish its long-running tightly coupled in-situ processing. In summary, our contributions in this paper are threefold.

- By modeling the in-situ processing, we present an algorithm for data analysis placement decision-making, and discuss errors in the decision-making. We further present algorithms with associated heuristic strategies to estimate key parameters and decrease the decision-making errors.
- We present the architecture and the implementation of a system that can enable existing data staging services to do the dynamic analysis placement decision-making online.
- We evaluate the presented online data analysis placement mechanisms for both synthetic and real analysis at large scales using different configurations. Our results show that the online data analysis placement mechanism can save up to 25% of in-situ processing time compared to predetermined analysis placement for real data analysis.

The rest of the paper is organized as follows. Section II presents background and motivation. Section III presents online data analysis placement policies. It also describes the architecture and the implementation to achieve the data analysis placement mechanisms. In Section IV, we evaluate presented solutions using synthetic and real data analysis. Related work is discussed in Section V. Finally, we conclude the paper and discuss future work in Section VI.

II. Background and motivation

In this section, we first introduce several hybrid in-situ processing workflows. By analysing their execution processes, we discuss the role of data analysis placement in workflow design. Next, we discuss how the solution based on an in-situ trigger performs online configuration adjustment during workflow execution. Finally, we use an example to show how analysis placement facilitates workflow performance.

A. Hybrid in-situ processing approaches

1) Hybrid in-situ processing with data reduction: This in-situ processing pipeline is divided into two stages. The first stage is usually a lightweight task that can reduce the size of the simulated data [11], and the second stage includes ana/vis which may need dedicated resource [18] or work on a refactored data structure [15]. The first stage is processed in a tightly coupled way because of its lightweight nature, and the second stage is executed in a loosely coupled way based on specialized data staging services.

2) Hybrid in-situ processing with optional task executions: When the simulated data contains interesting patterns warranting in-situ analysis, different stages of in-situ processing can be executed based on the content of the simulated data. Based on the detection results of the first stage [3], the task of the second stage is only executed when upon the appearance of interesting patterns. In particular, the interesting patterns can be the situation when a critical metric value locates in a specific range [17], or the detection of a topology property [2].

3) Generalized hybrid in-situ processing: The generalized hybrid in-situ processing mixes loosely coupled and tightly coupled paradigms with few assumptions about data analysis placement and execution. We can not properly determine the correct design before starting the workflow for the lack of prior knowledge regarding the simulation output or ana/vis. The placement of ana/vis can be decided by user-defined policies [6] during the workflow executions and supported by an in-situ tool that executes the same task either in a loosely or tightly coupled way [23]. Trigger primitives [10] can indicate the conditions to execute in-situ ana/vis.

B. Trigger adaptations for in-situ processing

Multiple works [3], [6], [10] adopt trigger mechanism to support adaptation for in-situ processing. The description of trigger is constructed by *detection*, *condition*, and *actions*. In particular, *domain-specific trigger* can efficiently support hybrid in-situ processing with optional ana/vis execution. The detection operation may use domain knowledge to detect the interesting phenomenons in the simulated data. The condition of the trigger relates the trigger actions with the trigger detection; the action of the trigger specifies the ana/vis task that is executed when the condition is satisfied.

A *domain-agnostic trigger* focuses on more general aspects of in-situ processing, especially for the workflow design configurations with multiple options. In addition to the data analysis placement options discussed in this work, the data processing frequency [12] and the computing resource [19] of in-situ processing can also be adjusted based on triggers during workflow execution. For example, the trigger action for data analysis placement adaption can be an indicator parameter that shows the data analysis placement patterns. The trigger detection and condition may adopt user-defined policies to specify which option is preferred in which situation based on workflow runtime information.

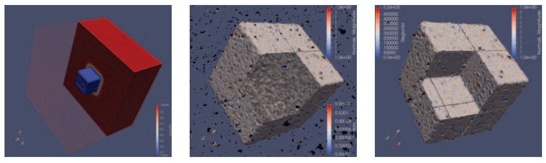
C. Requirements for adaptive data analysis placement

With changing simulation data across iterations, the execution time of the ana/vis may change in various ways. For example, we use the Gray-Scott simulation[4] as the data source shown in Figure 1 (a), then process the data generated by each

[2]We mainly use data analysis as an example of in-situ processing in this work, and our method can also be adapted to data visualizations.

[3]We consider the workflow execution time when deciding the data analysis placement in this work. The decision-making process can also be influenced by other factors such as resource efficiency and programming flexibility.

[4]https://github.com/pnorbert/adiosvm/tree/master/Tutorial/gray-scott



(a) Initial data (b) Isosurface (c) The largest isosurface area

Fig. 1. Visualization of data processing for the Gray-Scott simulation. Sub-figure (a) shows the initial data, Sub-figure (b) shows the results after isosurface extraction. Sub-figure (c) shows the largest isosurface area.

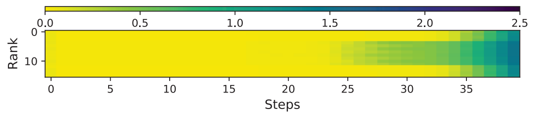


Fig. 2. Visualization of in-situ data analysis execution time. Darker colors represent a longer execution time.

simulation iteration. In particular, the data blocks are processed by isosurface filter shown in Figure 1 (b), then the largest isosurface area illustrated in Figure 1 (c) is extracted. If we use 16 ranks to generate 128MB of data for each computation iteration then run the data processing pipeline in the tightly coupled way. The in-situ processing time of different processes is visualized in Figure 2. One important observation is that the data analysis time varies across steps for every single rank and across ranks for the same step. These variations between steps and ranks showcase an opportunity to adopt different analysis placement strategies to reduce workflow execution time. For example, if the overhead of transferring data to a staging service is less than 2.0 sec, the simulation program can start the next iteration with less waiting time by processing the data analysis in the loosely coupled manner.

However, if we transfer a large number of data blocks and trigger an excessive number of threads in the staging area for loosely coupled processing, even if the overhead on the simulation can decrease, it is possible to overload the staging service, resulting in an increase in the execution time to finish in-situ processing. Therefore, a trade-off exists between the tightly coupled and the loosely coupled approach for data and associated analysis placement strategies during the workflow execution. Monitoring necessary information and adjusting the analysis placement strategy online is therefore a promising approach to reduce workflow execution time compared to a predetermined analysis placement strategy.

III. Adaptive data analysis placement policies

In this section, we model the workflow execution time for both the tightly coupled and the loosely coupled in-situ processing. Based on this model, we present a data analysis placement decision algorithm and discuss the types and reasons for decision-making errors. We also discuss how to collect the necessary information and estimate parameters required by the data analysis placement algorithm to decrease decision-making errors. We further discuss solutions based on heuristic strategies to decrease the decision errors and improve the workflow execution efficiency.

TABLE I
Major notations used in mathematical modeling

S	Simulation computation time for one iteration
A_t	Tightly coupled data analysis execution time
A_l	Loosely coupled data analysis execution time
T	Data transfer time from the simulation to the staging service
W_l	Waiting time to schedule the loosely coupled analysis

A. *Modeling of data analysis placement*

1) A motivating scenario: We assume the simulation computation contains two iterations (step 0 and step 1), and the in-situ processing is carried out at every iteration. In order to choose a suitable data analysis placement strategy, we need to compare the remaining time of the workflow execution after the first simulation iteration, which can be modeled by equation 1. Table I summarized the notations used for our mathematical modeling.

$$\begin{aligned} RW_t^0 &= A_t^0 + S^1 + A_t^1, \\ RW_l^0 &= T^0 + \max\{S^1 + A_t^1, W_l^0 + A_l^0\} \end{aligned} \tag{1}$$

The RW_t^0 (RW_l^0) represents the *remaining execution time of the workflow after the first simulation computation step with the tightly (loosely) coupled in-situ processing pattern*. For the tightly coupled case (RW_t^0), we process the data generated by the first iteration (step 0) of the simulation program, which takes A_t^0, then start the next simulation iteration (S^1). For the data generated by the next simulation computation (step 1), it is unnecessary to transfer data to staging service when there is no subsequent simulation computation; the tightly coupled approach (A_t^1) is adopted in this case. In contrast, the loosely coupled approach (RW_l^0) transfers the data generated by the first iteration to a staging service, which takes T^0, then starts the next iteration. The analysis in the staging service takes W_l^0 to schedule[5] a thread to execute the analysis, then it takes A_l^0 to finish the execution; however, since there is an overlap between the loosely coupled analysis and the next simulation iteration, the completion time of the full program is influenced by the maximum value between $S^1 + A_t^1$ and $W_l^0 + A_l^0$. Furthermore, we can use RS^0 to represent the term $S^1 + A_t^1$, which denotes the *remaining execution time of the simulation program after the step zero in-situ processing*. Since the RS^0 is the common term between RW_t^0 and RW_l^0, equation 1 can be reorganized as follows:

$$\begin{aligned} RW_t^0 &= A_t^0 + RS^0, \\ RW_l^0 &= T^0 + \max\{RS^0, W_l^0 + A_l^0\} \end{aligned} \tag{2}$$

Using RS as the x axis and RW as the y axis[6], we can further illustrate equation 2 by Figure 3 (a), which shows how the workflow execution time is affected by RS. In particular, the black color line represents the RW_l, when $RS < W_l + A_l$, RW_l equals to $T + W_l + A_l$, which is not affected by RS; the lines with other colors represents several possible RW_t values, which are affected by value of A_t; A_{ta}, A_{tb} and A_{tc} represents the typical execution time of the tightly

[5]The time between the reception of the RPC in the staging service and the moment at which the analysis thread actually runs in the staging service.

[6]We omit the superscript that represents the step number for the simplicity of visualization.

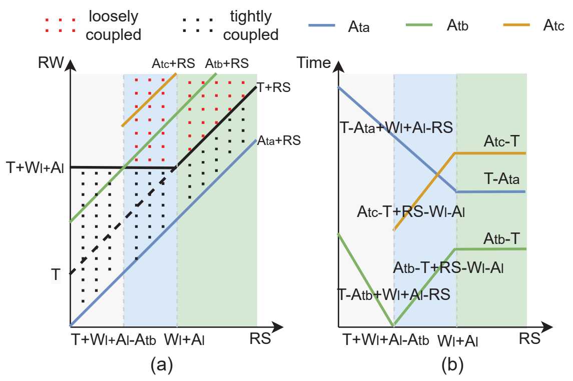


Fig. 3. Sub-figure (a) illustrates how the data analysis placement decision is influenced by different parameters. Sub-figure (b) shows the time saved by making the proper decision.

coupled analysis, respectively. Furthermore, Figure 3 (a) also visualizes the regions of preferable analysis placement pattern. If RW_t locates in the red dotted region ($RW_l < RW_t$), the loosely coupled pattern is preferable; if RW_t falls into the black dotted region ($RW_t < RW_l$), the tightly coupled pattern is preferable. According to the value range of RS, we can identify three sub-domains (shown in grey, blue and green color, respectively), and the strategies to choose a preferable data analysis placement pattern are described as follows:

1) If $RS \geq W_l + A_l$, when $A_t \geq T$, the loosely coupled approach is preferred; otherwise, the tightly coupled approach is preferred.
2) If $T + W_l + A_l - A_t \leq RS < W_l + A_l$, when $A_t + RS \geq T + W_l + A_l$, the loosely coupled approach is preferred, otherwise, the tightly coupled approach is preferred.
3) If $RS < T + W_l + A_l - A_t$, the loosely coupled approach becomes ineffective, and we always adopt the tightly coupled approach to run analysis.

We also visualize the benefits of adopting the preferable data analysis placement decision in Figure 3 (b). For example, for the line $A_{ta} + RS$ shown in Figure 3 (a), the preferable data analysis placement strategy is the tightly coupled pattern. Compared with adopting a loosely coupled pattern, it can save $T - A_{ta}$ when there is $RS > W_l + A_l$, and save $T + W_l + A_l - (A_{ta} + RS)$ when there is $RS \leq W_l + A_l$. If we separate the second part of the performance improvement into $T - A_{ta}$ and $W_l + A_l - RS$, it is clearer to show where the advantage comes from. In particular, $T - A_{ta}$ represents the reduction of the overhead of the simulation program when processing in a tightly coupled fashion (A_{ta}) as compared to transferring it to the staging area (T). The data staging service needs $W_l + A_l$ to finish the in-situ processing, which is longer than the remaining simulation process execution time (RS). Therefore, the $W_l + A_l - RS$ represents the reduction of the workflow completion time if we adopted a proper analysis placement strategy. Similarly, there are different performance improvements with the changing of preferred data analysis placement options depending on the different values of At, such as A_{tb} and A_{tc} shown in the Figure 3 (b).

2) An algorithm for data analysis placement decision-making: For a workflow that contains multiple simulation

Algorithm 1: Data analysis placement decision based on workflow runtime information.

```
1  $CouplingMethod$ = $Tight$;
2  if $step == 2$ then
3      $CouplingMethod$= $Loose$;
4  else if $step > 2$ then
5      $CouplingMethod$= $Tight$;
6      $RS, A_t, A_l, T, W$ = GetAndEstimateParameters();
7      if $RS > W_l + A_l$ then
8          if $A_t > T$ then
9              $CouplingMethod$=$Loose$;
10     else
11         if $A_t + RS > T + W_l + A_l$ then
12             $CouplingMethod$=$Loose$;
13     end
14 return $CouplingMethod$;
```

iterations, we can extend the model presented in Figure 3 (a) with minor modifications. Assuming that there is a barrier at the beginning of the simulation iteration, different processes can start at the same time when making the data analysis placement decision. We can reinterpret and extend equation 2 to determine the preferable analysis placement pattern of in-situ processing in any step. For the overhead of in-situ processing after i^{th} computation, if we adopt the tightly coupled pattern for this step (RW_t^i), it takes A_t^i to process the data generated at i^{th} step and then takes RS^i to finish simulation program. If we adopt the loosely coupled pattern for this step (RW_l^i), it takes T^i to transfer the data. In this case, the workflow completion time depends on the maximal value between the RS^i and the finish time of in-staging processing ($W_l^i + A_l^i$) for i^{th} step. Therefore, equation 2 can be viewed as a particular case where the i equals 0.

If we reuse the conclusion shown in Figure 3 (a) for deciding analysis placement pattern at any step. The process of dynamically making data analysis placement decisions can be further described by Algorithm 1. In this algorithm, we set variable *CouplingMethod* to *Loose* or *Tight* to represents corresponding data analysis placement decisions. We force the data analysis to be processed in a tightly coupled manner in the first two steps[7] and in a loosely coupled manner for the third step of the in-situ processing. This allows us to have initial data records for all parameters used in the decision-making process. *GetAndEstimateParameters()* function, which is described in subsection III-B, can access and estimate the key parameters used in decision-making. Depending on the value of RS, we use different conditions to select the proper data analysis placement pattern, and these conditions come from the results shown in Figure 3 (a). It is worth noting that Algorithm 1 can be executed by simulation processes separately, and every process may adopt different data analysis placement decisions. Subsection III-D introduces more details about how the analysis placement decision-making of one

[7]The first step starts from zero, and it may contain initialization that influences the analysis execution time.

TABLE II
DECISION ERROR TYPES AND REASONS

Error types	Reasons			
Tightly→Loosely	$\uparrow A_t$	$\downarrow T$	$\uparrow (A_t + RS)$	$\downarrow (T + W_l + A_l)$
Loosely→Tightly	$\downarrow A_t$	$\uparrow T$	$\downarrow (A_t + RS)$	$\uparrow (T + W_l + A_l)$

process is affected by other processes.

3) Error of the decision-making: In Algorithm 1, we use *GetAndEstimateParameters()* to estimate the key parameters for decision-making based on collected historical values. It is necessary to discuss how errors in estimating these parameters influence the decision results made by Algorithm 1.

The first column of Table II lists two typical errors for decision-making. In particular, the "Tightly→Loosely" (abbreviated as TL error) represents the case that the tightly coupled pattern is misclassified as the loosely coupled pattern. Conversely, the "Loosely→Tightly" (abbreviated as LT error) means we mispredict and perform tightly-coupling instead of processing it in a loosely coupled way. According to line 8 and line 11 in Algorithm 1, the TL error happens if there is a higher estimation of the left-hand side term or a lower estimation of the right-hand side term. In contrast, a lower estimation of the left-hand side term or a higher estimation of the right-hand side term may cause the LT error. We summarize these reasons in Table II. For example, $\uparrow A_t$ represents a higher estimation of A_t compared with its real value. The real value might be less than T, and the tightly coupled pattern should be adopted, but we misclassified it as the loosely coupled pattern. When decision error occurs, Algorithm 1 cannot achieve the benefits illustrated in Figure 3 (b). Subsection III-B and subsection III-C further illustrate typical scenarios that cause errors in decision-making and how our solutions eliminate errors through online parameter estimation techniques.

B. Parameter collection and estimation

From line 6 of Algorithm 1, we need to estimate the following parameters for making the analysis placement decision: the remaining execution time of the simulation program after the current in-situ processing step (RS); the future data analysis execution time when executed in a tightly coupled and loosely coupled manner (A_t and A_l); the future data transfer time (T) and the wait time (W_l) to schedule computing resources in the staging service. However, what we can measure directly are the latest A_l and W_l from the data staging service, and A_t, S, and T for the previous in-situ processing steps from the simulation process. Algorithm 2 provides details to estimate the future value of these key parameters based on the collected metrics during the workflow execution.

From line 1 to line 2 of Algorithm 2, we acquire the key parameters from a metric store[8] integrated with the simulation process and staging service. For the *naive* strategy in line 3, it returns the measured parameters directly. However, this strategy may introduce inaccurate parameter estimations and multiple decision-making errors. In particular, RS in Algorithm 1 represents the remaining execution time of the simulation program after the current in-situ processing step. However, the naive strategy in Algorithm 2 returns the computation time for one iteration step, which leads to the RS value being smaller than the actual situation with multiple computation steps. Besides, we only execute the data analysis either in a loosely coupled or tightly coupled way, if the last decision is the loosely coupled pattern, there is no latest record of A_t; on the contrary, the recorded A_l value is outdated, which may also lead to inaccurate parameter estimations.

[8]The infrastructure to collect key parameters is discussed in Section III-E.

Algorithm 2: Getting and estimating the key parameters during the workflow execution.

```
1  $A_t, S, T$ = getLatestMetricsSim();
2  $A_l, W_l$ = getLatestMetricsStage();
3  **if** *strategy==“naive”* **then**
4      return $S, A_t, A_l, T, W$
5  // *For the strategy based on the parameter estimation*
6  $EA_l = A_l$, $EA_t = A_t$;
7  **if** *lastDecision==“tightly”* **then**
8      // *Updating records about loosely coupled analysis*
9      **if** $A_l < A_t$ **then**
10         $EA_l = A_t$;
11     **if** $A_l > A_t$ *and stageIsIdle* **then**
12         $EA_l = A_t$;
13     $P = A_t$;
14 **else**
15     // *Updating records about tightly coupled analysis*
16     **if** $A_l < A_t$ *and* $A_l \neq 0$ **then**
17         $EA_t = A_l$;
18     **if** $A_l > A_t$ *and stageIsIdle* **then**
19         $EA_t = A_l$;
20     $P = T$;
21 **end**
22 Updating $T_{avg}$, $S_{avg}$ and $P_{avg}$ by Iterative Averaging;
23 $Rstep = totalSimStep - currentSimStep$;
24 $ERs = Rstep \times (S_{avg} + Freq \times P_{avg})$;
25 **return** $ERs, EA_t, EA_l, T_{avg}, W_l$
```

In order to improve the accuracy of the parameter estimations, we need to utilize the collected information in a more efficient way. Since the computing resources used for the in-staging service are usually less than the simulation program, executing the same analysis function in the loosely coupled pattern may not achieve a faster execution time compared with executing it in the tightly coupled pattern. Therefore, when there is $A_l < A_t$, we can confirm that one of them is outdated. At line 9 and line 16 in Algorithm 2, we estimate associated outdated values according to the decision of the last step. In contrast, when $A_l > A_t$, we only update the outdated value if the staging service is comparatively idle and light loaded, such as line 11 and line 18 in Algorithm 2.

When numerous threads saturate the data staging service, it needs far more time to finish the data analysis execution in a loosely coupled pattern compared with the tightly coupled pattern. If we update the A_t based on the latest A_l value, it may cause an estimation of A_t to be higher than the actual value.

In order to detect the load of the staging service, we define the metric *staging service load* by the number of running threads divided by the number of available threads in the data staging service. The analysis placement procedure can acquire the latest metric value from the staging service before deciding the placement pattern. We classify the staging load by two threshold values, which is a classical dynamic load balancing strategy [14]: if the stage load is less than a lower threshold, it is *under-loaded*; if the stage load is larger than a high threshold, it is *over-loaded*; the staging service is *normal-loaded* when the stage load locates between two thresholds. We assume the stage service is idle when it is under-loaded. Line 11 and line 18 only guarantee there is high confidence estimation of A_l or A_t value when the staging service is under-loaded. When the staging service is not under-loaded, we can not use the $A_t(A_l)$ to estimate $A_l(A_t)$ because of the interference of the high staging load. We address these edge cases in subsection III-C based on a heuristic strategy.

At line 22 in Algorithm 2, we update the average data transfer time (T_{avg}) and simulation computation time (S_{avg}) based on the latest measurements. We also compute the average time to do the in-situ processing from the perspective of the simulation program (P_{avg}); the in-situ processing time at the current iteration is determined by the analysis placement pattern (line 13 and line 20). Based on the estimated A_t and A_l values, we can estimate the remaining simulation execution time if there is prior knowledge about the total number of steps of the simulation and the frequency of in-situ processing. At line 24 in Algorithm 2, we estimate the remaining simulation execution time based on average simulation computation time and in-situ processing overhead on the simulation process. However, the current estimation of the remaining time only works well for simulations in which there is a small difference (white noise) in computation time between different iterations. Otherwise, we may need prior knowledge, such as the distribution of the simulation computation time, when estimating the remaining execution time of the simulation. Another assumption behind the current estimation algorithm is the gradual and smooth change for key parameters between two adjacent decision-making steps. The frequent fluctuations of key parameters may cause more false attempts and decrease the efficiency of the adaptive data analysis placement mechanism.

C. Adjusting decisions for edge cases

At line 11 and line 18 in Algorithm 2, we update $A_l(A_t)$ value when the staging service is idle (under-loaded). If the staging service is normal-loaded or over-loaded, $A_l > A_t$ is caused by either the performance degradation of the staging service or the actual increase of the data analysis execution time. We still fail to estimate the $A_l(A_t)$ because of the difficulty to accurately identify how the high staging load influences data analysis execution time. We present a heuristic strategy to decide the analysis placement pattern.

Heuristics 1: When the staging service is neither over-loaded nor under-loaded, for the process that adopts the tightly coupled pattern in the current step, and there is $A_l(A_t) > T$, we reset them to the loosely coupled pattern.

The aforementioned heuristics is applied after Algorithm 1 finishes. In particular, when the staging service is over-loaded, we have high confidence that there is service performance degradation, Algorithm 1 can guarantee that the proper analysis placement is a tightly coupled pattern. When the staging service is normal-loaded, we may underestimate A_t or overestimate the A_l with the fact that $A_l > A_t$. As a consequence, some "Loosely$\rightarrow$Tightly" errors exists based on decision error reasons listed in Table II. In order to decreases the possibility of these errors, we adjust placement decision to the tightly coupled pattern for cases where the data analysis execution time may longer than data transfer time.

D. Adjusting decisions from collective perspective

In previous subsections, we mainly discussed the data analysis placement decision-making mechanism across different steps for one simulation process. However, the suitable decision for one process may not be the proper one from the perspective of multiple processes. For example, it is common for simulation programs to use barriers at the beginning of each iteration for the correctness of parallel computation. If one process needs a much longer time to finish the in-situ analysis than the other processes, and this process happens to place the data analysis in a tightly coupled manner, it can cause unnecessary wait for other simulation processes.

Heuristics 2: When the staging service is not over-loaded, we compute the average analysis execution time based on the latest records (both for A_t and A_l). If the data analysis placement decision is a tightly coupled pattern and the current A_t or A_l value exceeds a threshold, such as the average analysis execution time, we reset the data analysis placement decision to the loosely coupled pattern.

We use *Heuristics 2* to identify tightly coupled analysis with a long execution time compared with the average execution time, then reset them as the loosely coupled pattern. In this way, the simulation program will not be blocked by a few long-running analysis and can quickly move to the next iteration.

E. Implementation overview

Figure 4 illustrates the architecture to support the adaptive data analysis placement decision algorithms. To implement loosely coupled placement strategy, we use the existing data staging service [21] that is built with the Mochi framework [16] to store and process the data based on RPC and RDMA. It can store the data block in the memory of the staging service and triggers the associated data analysis running by user-level threads. Every simulation process binds to a staging process at the beginning of the workflow. The data block is transferred to the corresponding staging process. The *Metric data store* is a map that stores metric values under different keys, and metric values are stored in a circular buffer since we only use several latest measurements to decide the analysis placement pattern in Algorithm 1. The *Metric estimation* component in the staging service is responsible for

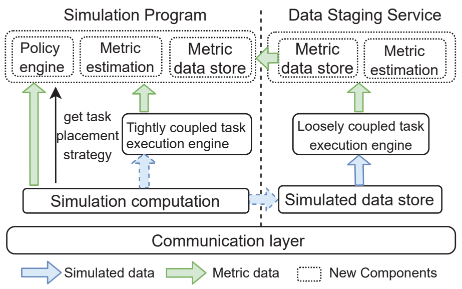


Fig. 4. The architecture of the online monitor and decision-making service for in-situ processing.

collecting metrics such as data analysis execution time and waiting time to schedule the data analysis during the workflow execution. It also records the number of active threads in order to calculate the load of the staging service. Every simulation process can access this information from the associated data staging service by *getStageStatus* RPC.

Before each in-situ processing step, the simulation process calls the *getStageStatus* RPC to get the metrics stored in the data staging service. Combined with metrics recorded in *Metric data store*, such as simulation computation time, data transfer time, and tightly coupled data analysis execution time, the *Policy engine* adopts Algorithm 1 to make the data analysis placement decisions. The *Metric estimation* is in charge of estimating key parameters for data analysis placement decision-making based on the mechanism discussed in subsection III-B, III-C and III-D. If the data analysis placement decision is the tightly coupled pattern, the data analysis is scheduled to run on the same computing resource as the simulation after each simulation time-step. For the loosely coupled decision, the data block is transferred to the staging service and executed in the staging area asynchronously. The simulation moves to the next iteration when all processes finish in-situ processing.

IV. Evaluation

A. Platform and applications

The evaluation in this work is executed on the Haswell partition of NERSC Cori System [13]. **The code and scripts used for evaluation are publicly available.**[9] The workflow used for evaluation contains several components: the mini-simulation[10] programmed with MPI; the real-life data analysis and synthetic analysis[11]; the in-situ library that can process the data analysis in a tightly coupled way; the data staging service that processes the data analysis in the loosely coupled fashion on separate nodes, and components that store and estimate the key parameters for adaptive analysis placement decision-making. In this workflow, the data produced by simulation processes is analyzed in each iteration and the output of each analysis step are several orders of magnitude smaller than the simulation output. We compare multiple data analysis placement strategies and their impact on the total workflow execution time. For ***Tightly*** pattern, we process all data for every in-situ processing step in a tightly coupled fashion. For ***Loosely*** pattern, we process all data for every in-situ processing step in a loosely coupled manner. Instead of fixing the data analysis placement pattern before running the workflow, dynamic strategies update it during the workflow execution. For ***Dynamic-naive*** pattern, we use the latest parameter values collected during the workflow execution as the input for policy to decide the data analysis placement strategy. In contrast, ***Dynamic-estimation*** pattern uses the parameter estimation discussed in Algorithm 2 as the input for data analysis placement decision. ***Dynamic-estimation-heuristics*** adopts both parameter estimation and the heuristics discussed in subsection III-C and subsection III-D to adjust the data analysis placement decisions. For the experiments in subsection IV-B, we use synthetic analysis to compare the efficiency of the dynamic data analysis placement strategies. We configure the analysis execution time to construct various scenarios with variable key parameters during the workflow execution. Furthermore, experiments in subsection IV-C evaluate the benefits and limitations of the dynamic data analysis placement strategies in more diverse and large-scale cases for real-life data analysis.

[9]https://git.io/JZhja

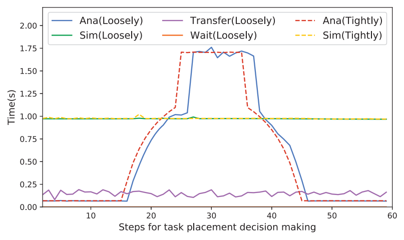
[10]https://github.com/pnorbert/adiosvm/tree/master/Tutorial/gray-scott

[11]The synthetic analysis is constructed by the for loop with a configurable iteration number. In each iteration, it sleeps 1ms and generates random double values to fill in a vector. Then we add values stored in the vector together.

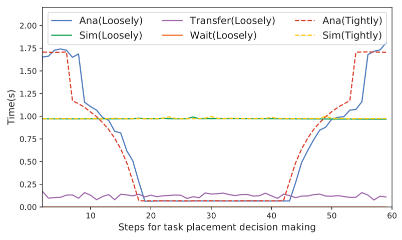
B. Experiments with synthetic data analysis

In this experiment, we aim to show the efficiency of the adaptive data analysis placement strategy for several typical cases adopted by in-situ processing. There are 32 processes (4 physical cores for each process) for the simulation and 8 processes (8 physical cores for each process) for the data staging service. We choose the aforementioned process number to guarantee there is enough memory to process the simulated data. The simulation iterates 60 steps in total, and there is 1GB of data generated by the simulation in each step. We extract several representative cases from the workflow that contains real-life data analysis, and these cases pave the way for evaluating the workflow that contains more complicated data analysis (discussed in subsection IV-C). For Case 1 shown in Figure 5(a), we run the workflow by *Loosely* and *Tightly* pattern separately and illustrate collected metric values in the figure. In particular, the analysis starts from a comparatively short execution time, increases to a large value, and then falls back to a small value. This case imitates the situation when interesting data appear gradually, and we need more analysis time to process these data. In contrast, for Case 2 illustrated in Figure 5 (b), the analysis execution time starts from a comparatively large value and then decreases to a small value and again increases to the large level at last. This case imitates the situation when interesting data disappear for a while and appear gradually. For Case 3 shown in Figure 5 (c), the data analysis execution time fluctuates around the data transfer time periodically. This case imitates the situation when data analysis execution time is approximated to the data transfer time.

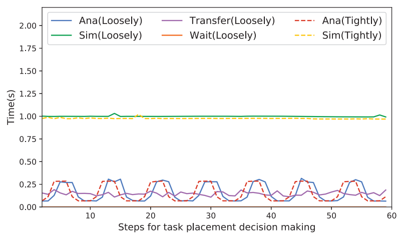
Figure 5 (d), Figure 5 (e) and Figure 5 (f) present the online data analysis placement decision results for three associated cases, respectively. Since different processes use the



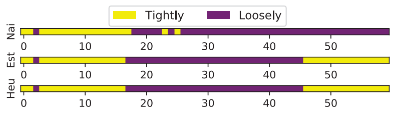
(a) Case 1



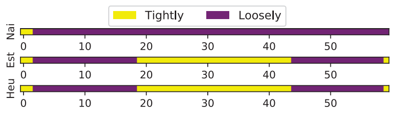
(b) Case 2



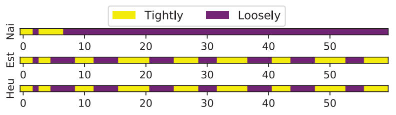
(c) Case 3



(d) Dynamic decisions for Case 1



(e) Dynamic decisions for Case 2



(f) Dynamic decisions for Case 3

Fig. 5. Sub-figure (a), (b) and (c) show key parameters during the workflow execution for different cases. Sub-figure (d), (e) and (f) visualize associated dynamic data analysis placement decisions of each in-situ processing for the rank 0 process. The “Nai”, “Est” and “Heu” represent the *Dynamic-naive*, *Dynamic-estimation*, and *Dynamic-estimation-heuristics*, respectively.

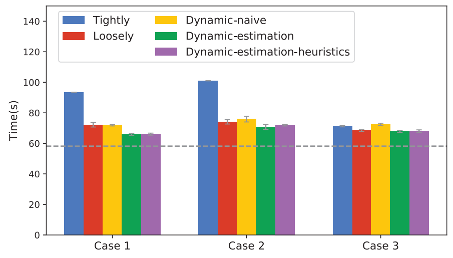
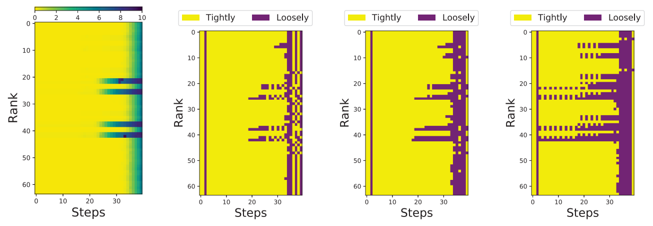


Fig. 6. The workflow execution time for the cases with a varied analysis execution time. The gray dash line labels the accumulated simulation computation time on average. The error bar represents the standard deviation.



(a) Analysis time (b) Naive (c) Estimation (d) Heuristics

Fig. 7. Sub-figure (a) indicates the data analysis execution time when the in-situ processing runs in *Tightly* pattern for different processes. The darker color represents a longer execution time. Sub-figure (b), Sub-figure (c), and Sub-figure (d) visualize the data analysis placement decisions based on the *Dynamic-naive*, *Dynamic-estimation*, and *Dynamic-estimation-heuristics* patterns, respectively.

same synthetic analysis, there are similar parameters between different ranks during the workflow execution. We use the process with rank equals 0 as a sample for data analysis placement decision visualization. Compared with *Dynamic-naive* pattern, *Dynamic-estimation* and *Dynamic-estimation-heuristics* pattern can better capture the changes of the workflow execution. In particular, for the decision results shown in Figure 5 (d), both the naive strategy and the estimation strategy can capture the change at the 16^{th} step, but the data is always outdated for the naive pattern, and it can not capture the change at the 45^{th} step. However, using the estimation-based strategies, we can properly detect the switch between decision changes. Similarly, for Figure 5 (e) and Figure 5 (f), the naive strategy can not detect the parameter change because of the inaccurate parameter estimations.

Figure 6 illustrates the workflow execution time associated with cases shown in Figure 5 for various data analysis placement strategies. In particular, for Case 1 and Case 2, the dynamic patterns show better performance than the *Tightly* pattern. Besides, the patterns that adopt parameter estimation can better detect the workflow changes than the naive strategy, leading to a shorter workflow execution time for both cases. However, for Case 3 shown in Figure 5, even if the dynamic patterns based on parameter estimation can capture the change of the workflow, the frequent switch between data analysis placement options also introduces many trial-and-error data analysis placement for collecting the latest data. There is not much time saving for dynamic patterns in this case since the overhead counteracts the benefits.

C. Experiments with actual data analysis

This experiment aims to evaluate the efficiency of dynamic data analysis placement strategies for actual in-situ data analysis. We use the data analysis pipeline presented in subsection II-C, which extracts the isosurface from simulated data and calculates the largest area. We use 64 MPI processes (2 cores for each process) for the simulation and 4 MPI processes (8 cores for each process) for the data staging services. We run the simulation for 40 iterations, and there is 2GB of data generated at each iteration. The data is processed after the simulation computation for each iteration.

Figure 7 (a) illustrates the data analysis time of different ranks across different data processing steps when there are 64 simulation processes. The load unbalances of data analysis execution time between different data analysis is observable. Figure 7 (b), Figure 7 (c) and Figure 7 (d) show decision results of different data analysis placement strategies, respectively. In particular, for *Dynamic-naive* pattern, there are frequent fluctuations when the data analysis execution time is comparatively large. This is caused by the outdated data of the key parameters. For *Dynamic-estimation* pattern, the decision region is more continuous, but there are some

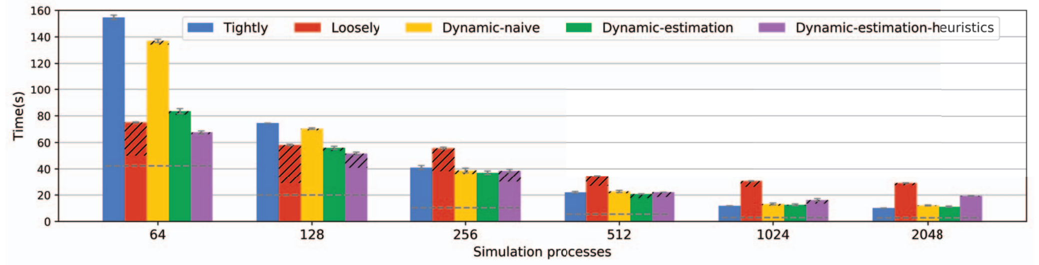


Fig. 8. The execution time of workflow using real data analysis. The horizontal axis represents different simulation process numbers. The gray dash line shows the accumulated time for simulation computation on average. The shadowed area represents the time spent on the staging area after the simulation program finishes. The error bar represents the standard deviation of execution time.

different discrete decisions around the 36th step. These inconsistent decisions cause other data analysis to wait for the unbalanced analysis with a longer execution time, such as the dark color region shown in Figure 7 (a), which causes unnecessary overhead. *Dynamic-estimation-heuristics* pattern adopts the heuristic strategies discussed in subsection III-C and subsection III-D to further decrease decision errors and the overhead to wait for long-running data analysis. The decisions after the 30th step are more consistent, and more data blocks and associated analysis are transferred into the staging service compared with other patterns based on dynamic strategies.

For evaluating the performance of the dynamic analysis placement strategies in different configurations, we increase the simulation process number from 64 to 2048 with the same number of data staging services and the same simulated domain size. The data block number increases and the data block size decrease with the increase of the process number. As illustrated in Figure 8, when there are 64 or 128 processes, *Loosely* pattern shows better execution time compared with *Tightly* pattern; this is because of the unbalanced data analysis that increases the simulation execution time for *Tightly* pattern. However, dumping all the data into the data staging area can overload the data staging processes. For example, when there are 256 processes, the execution time of *Loosely* pattern is slightly longer than *Tightly* pattern. This is because the overhead caused by saturating all staging resources exceeds its benefits. The dynamic analysis placement strategy tries to balance the analysis execution and available computing resources of the data staging service. In particular, compared with the pre-configured pattern that takes less execution time, such as *Loosely* pattern for 64 processes and 128 processes, and *Tightly* pattern for 256 processes, the in-situ processing time[12] decreases by around 25%, 19% and 12%, respectively.

With an increase in the number of simulation processes, such as 512 processes, the data analysis time decreases, and the *Tightly* pattern tends to be optimal. Our dynamic strategies based on parameter estimation can achieve a similar workflow execution time. If we further increase the simulation processes from 1024 to 2048, the simulation computation did not decrease anymore. With the continuing decrease of the block size and analysis execution time, most of the analysis are preferable to be processed by the *Tightly* pattern. However, with the increase of the simulation process number such as 1024 or 2048, the *Dynamic-estimation-heuristics* pattern shows extra overhead. This is because the heuristic strategy needs to do the collective operation to detect the long-running data analysis based on the average value; without prior knowledge about how the simulation and associated analysis execution time change across multiple steps, dynamic strategies can achieve performance improvements for all configurations as compared to a static approach. While one can run the workflow several times to determine its characteristics manually, it is preferable and more efficient to run the workflow once with a fairly small overhead to determine the placement of analysis adaptively.

[12]The workflow execution time minus the accumulated simulation computation time.

Further efforts include making the dynamic analysis placement strategies run in a more autonomic way. For example, we can select a proper dynamic strategy as needed; for the situation where the tightly coupled pattern is definitely preferable, and overhead of collective communication for heuristics exceeds its benefits (such as cases where simulation adopts 1024 or 2048 processes in Figure 8), we can avoid using the heuristic strategies.

V. Related work

Jin et al. [6] presented a framework that can dynamically change the data analysis placement strategy of in-situ processing based on adaptive policies. This work also uses collected workflow run-time information to guide the data analysis placement decision in a hybrid way. However, this work does not elaborate on how to process the situation where ana/vis varies across steps for every single rank and across different ranks for the same step. Besides, they did not discuss details about how to collect and estimate the key parameters used for ana/vis placement decision-making.

Tu et al. [5] and Wang et al. [21] presented models to compare tightly coupled and loosely coupled in-situ processing. Similarly, Kress et al. [7], [8] evaluated the tightly and loosely coupled in-situ visualization pipeline with varied configurations such as available resources, type of visualization algorithms, and visualization frequency, and they show each pattern are preferred in a particular configuration. However, they did not discuss how to use the model to guide the ana/vis placement strategy during the workflow execution.

Zhang et al. [23] presented FlexIO that aims at providing the flexibility of ana/vis placement among the I/O path for in-situ processing. This work focuses on the capability to place the in-situ ana/vis. Our work focuses on using similar capabilities in a more strategic way based on the workflow run-time information. Sun et al. [20] focus on adaptive data placement in the data staging area to decrease the data access time; however, our work focuses on the adaptive task placement for decreasing workflow execution time.

Another set of works model resource cost and utilization of in-situ processing. In particular, Kress et al. [9] explore how to chose proper node or core partition for different types of in-situ processing. Aupy et al. [1] presented a memory-constraint model to determine the resources used for the in-situ analysis. However, our work assumes the computing resource is fixed at the beginning of the workflow and tries to place the ana/vis dynamically to facilitate the workflow execution.

VI. Conclusion and Future work

In this paper, we demonstrated the advantage of adjusting the in-situ processing data analysis placement strategy dynamically to decrease workflow execution time. By modeling in-situ processing and developing online data analysis placement policies, we showed the benefits and limitations of presented policies in various experimental configurations. In future work, we plan to integrate the presented online data analysis placement mechanism into workflows that contain more complicated in-situ processing pipelines.

Acknowledgement

The research presented in this paper was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program. This research was also supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This work was also supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357.

References

[1] G. Aupy, B. Goglin, V. Honoré, and B. Raffin. Modeling high-throughput applications for in situ analytics. *The International Journal of High Performance Computing Applications*, 33(6):1185–1200, 2019.

[2] J. C. Bennett, H. Abbasi, P. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–9, 2012.

[3] J. C. Bennett, A. Bhagatwala, J. H. Chen, A. Pinar, M. Salloum, and C. Seshadhri. Trigger detection for adaptive scientific workflows using percentile sampling. *SIAM Journal on Scientific Computing*, 38(5):S240–S263, 2016.

[4] H. Childs, S. D. Ahern, J. Ahrens, A. C. Bauer, J. Bennett, E. W. Bethel, P.-T. Bremer, E. Brugger, J. Cottam, M. Dorier, et al. A terminology for in situ visualization and analysis systems. *The International Journal of High Performance Computing Applications*, 34(6):676–691, 2020.

[5] T. M. A. Do, L. Pottier, S. Thomas, R. F. da Silva, M. A. Cuendet, H. Weinstein, T. Estrada, M. Taufer, and E. Deelman. A novel metric to evaluate in situ workflows.

[6] T. Jin, F. Zhang, Q. Sun, M. Romanus, H. Bui, and M. Parashar. Towards autonomic data management for staging-based coupled scientific workflows. *Journal of Parallel and Distributed Computing*, 146:35 – 51, 2020.

[7] J. Kress, M. Larsen, J. Choi, M. Kim, M. Wolf, N. Podhorszki, S. Klasky, H. Childs, and D. Pugmire. Comparing the efficiency of in situ visualization paradigms at scale. In *International Conference on High Performance Computing*, pages 99–117. Springer, 2019.

[8] J. Kress, M. Larsen, J. Choi, M. Kim, M. Wolf, N. Podhorszki, S. Klasky, H. Childs, and D. Pugmire. Comparing time-to-solution for in situ visualization paradigms at scale. In *2020 IEEE 10th Symposium on Large Data Analysis and Visualization (LDAV)*, pages 22–26, 2020.

[9] J. Kress, M. Larsen, J. Choi, M. Kim, M. Wolf, N. Podhorszki, S. Klasky, H. Childs, and D. Pugmire. Opportunities for cost savings with in-transit visualization. In *International Conference on High Performance Computing*, pages 146–165. Springer, 2020.

[10] M. Larsen, A. Woods, N. Marsaglia, A. Biswas, S. Dutta, C. Harrison, and H. Childs. A flexible system for in situ triggers. In *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ISAV '18, pages 1–6, New York, NY, USA, 2018. ACM.

[11] S. Li, N. Marsaglia, C. Garth, J. Woodring, J. Clyne, and H. Childs. Data reduction techniques for simulation, visualization and data analysis. In *Computer Graphics Forum*, volume 37, pages 422–447. Wiley Online Library, 2018.

[12] P. Malakar, V. Vishwanath, C. Knight, T. Munson, and M. E. Papka. Optimal execution of co-analysis for large-scale molecular dynamics simulations. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 702–715, 2016.

[13] NERSC. The Cori System. https://docs.nersc.gov/systems/cori/.

[14] A. Osman and H. Ammar. Dynamic load balancing strategies for parallel computers. *Sci. Ann. Cuza Univ.*, 11:110–120, 2002.

[15] Z. Qiao, T. Lu, H. Luo, Q. Liu, S. Klasky, N. Podhorszki, and J. Wang. Sirius: Enabling progressive data exploration for extreme-scale scientific data. *IEEE Transactions on Multi-Scale Computing Systems*, 4(4):900–913, 2018.

[16] R. B. Ross, G. Amvrosiadis, P. Carns, C. D. Cranor, M. Dorier, K. Harms, G. Ganger, G. Gibson, S. K. Gutierrez, R. Latham, et al. Mochi: Composing data services for high-performance computing environments. *Journal of Computer Science and Technology*, 35(1):121–144, 2020.

[17] M. Salloum, J. C. Bennett, A. Pinar, A. Bhagatwala, and J. H. Chen. Enabling adaptive scientific workflows via trigger detection. In *Proceedings of the first workshop on in situ infrastructures for enabling extreme-scale analysis and visualization*, pages 41–45, 2015.

[18] C. Sewell, K. Heitmann, H. Finkel, G. Zagaris, S. T. Parete-Koon, P. K. Fasel, A. Pope, N. Frontiere, L. Lo, B. Messer, S. Habib, and J. Ahrens. Large-scale compute-intensive analysis via a combined in-situ and co-scheduling workflow approach. In *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2015.

[19] A. Shashidharan, R. R. Vatsavai, and R. K. Meentemeyer. Futures-dpe: towards dynamic provisioning and execution of geosimulations in hpc environments. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 464–467, 2018.

[20] Q. Sun, T. Jin, M. Romanus, H. Bui, F. Zhang, H. Yu, H. Kolla, S. Klasky, J. Chen, and M. Parashar. Adaptive data placement for staging-based coupled scientific workflows. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2015.

[21] Z. Wang, P. Subedi, M. Dorier, P. E. Davis, and M. Parashar. Staging based task execution for data-driven, in-situ scientific workflows. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 209–220, 2020.

[22] F. Zheng, H. Abbasi, J. Cao, J. Dayal, K. Schwan, M. Wolf, S. Klasky, and N. Podhorszki. In-situ i/o processing: a case for location flexibility. In *Proceedings of the sixth workshop on Parallel Data Storage*, pages 37–42, 2011.

[23] F. Zheng, H. Zou, J. Cao, J. Dayal, T. Nugye, G. Eisenhauer, and S. Klasky. Flexio: Location-flexible execution of in situ data analytics for large scale scientific applications. In *Proc. IEEE International Parallel and Distributed Processing Symp (IPDPSâ AZ13)*, pages 320–331, 2013.