Modeling High-throughput Applications for in situ Analytics

Guillaume Aupy, Brice Goglin, Valentin Honoré, Bruno Raffin

Modeling High-throughput Applications for *in situ* Analytics

Guillaume Aupy[1], Brice Goglin[1], Valentin Honoré[1], and Bruno Raffin[2]

[1]Inria, LaBRI, Univ. Bordeaux, 33400 Talence, France
[2]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France

April 5, 2019

Abstract

With the goal of performing exascale computing, the importance of I/O management becomes more and more critical to maintain system performance. While the computing capacities of machines are getting higher, the I/O capabilities of systems do not increase as fast. We are able to generate more data but unable to manage them efficiently due to variability of I/O performance. Limiting the requests to the Parallel File System (PFS) becomes necessary. To address this issue, new strategies are being developed such as online *in situ* analysis. The idea is to overcome the limitations of basic *post-mortem* data analysis where the data have to be stored on PFS first and processed later. There are several software solutions that allow users to specifically dedicate nodes for analysis of data and distribute the computation tasks over different sets of nodes. Thus far, they rely on a manual resource partitioning and allocation by the user of tasks (simulations, analysis).

In this work, we propose a memory-constraint modelization for *in situ* analysis. We use this model to provide different scheduling policies to determine both the number of resources that should be dedicated to analysis functions, and that schedule efficiently these functions. We evaluate them and show the importance of considering memory constraints in the model. Finally, we discuss the different challenges that have to be addressed in order to build automatic tools for *in situ* analytics.

Keywords: *in situ* analysis, *in transit* analysis, resource partitioning, scheduling algorithms

Contents

1 Introduction

High Performance Computing (HPC) systems are very large computing systems (for instance Summit at ORNL has two million cores). Due to their scale, new problems such as data management, fault tolerance or large-scale scheduling arise and force the community to design new solutions for those systems.

HPC applications are typically composed of two parts: simulations, that generate data, and analysis, that post-process these data to generate a final output for the application. As an example, a climate model simulation will generate temporal images of a climate evolution. Then later this data can be visualized and analyzed for scientific discovery.

There are two main paradigms to link simulation and analysis. The first one, called *out-of-machine*, consists in performing all the simulation on a machine A and perform the analysis on a machine B. The strength of this paradigm is that a full machine is dedicated to simulation, which leads to good computing performance. However, the trade-off is that all the simulation data has to be stored (using for instance the Parallel File System, PFS) to allow machine B to analyze it later. The increase in computing power enables the current simulation codes to generate increasingly large amounts of data. Meanwhile, the I/O performance of HPC systems does not suffice to handle these data at the same rate. Thus, bottlenecks appear due to the I/O system that severely impairs the application performance, reducing the increase in computing power.

To limit I/Os and the associated overheads, a second paradigm called *in situ* proposes to perform the analysis and simulation on the same machine without relying on intermediate files. Analysis is performed on-line, starting as soon as the data produced by the simulation are available in the memory of the compute nodes. Only the output of analysis functions are written to the PFS. Thus, the simulation and analysis share the same computing resources. The challenge is then to optimize resource allocation between the analysis and the simulation. Today's *in situ* processing frameworks mainly rely on the user to perform this resource allocation for each application and each platform.

The goal of this work is to provide efficient algorithms to perform both resource partitioning and analysis allocations. In this work we make the following contributions:

- We propose a general model for HPC applications that take into account simulations and analysis functions as well as a model for the target machine, including multi-core architectures, memory limitations, communication costs under bandwidth constraints. We use it to properly define the *in situ* allocation problem.
- We use this model to derive new algorithms for the allocation and schedul-

ing problem. The algorithms are based on a theoretical analysis of the model followed by greedy strategies. We also provide an optimal (non polynomial) solution which will be used later to study our strategies.

- We evaluate these algorithms on synthetic applications. This evaluation allows us to point out the key elements to take into account when scheduling *in situ* functions.

Note that the model that we study supports mixed strategies for the analysis scheduling. The analysis can be performed on the same nodes that run the simulation, either in sequence with the simulation execution or in overlap following a time or space sharing strategy. These strategies are commonly referred as *in situ*. But they can also be performed *in transit*, i.e. on nodes dedicated to the analysis taking into account the extra cost of the data transfer from the simulation nodes to these staging nodes.

The rest of the paper is organized as follows. Section 2 is dedicated to related work. In Section 3, we formalize our application models and platform features, before introducing the resource partitioning problem in Section 4. In Section 5, we describe some scheduling policies for analysis tasks in *in situ* paradigm. Section 6 is dedicated to present and discuss simulation results. Finally, we provide concluding remarks and future directions in Section 7.

2 Related Work

Solutions to *in situ* visualization and analysis are usually described by where the process is performed and how resources are shared. This has led to the common distinction between *in situ*, *in transit*, and postmortem visualization, as well as between tight and loose coupling. To alleviate the I/O bottleneck, the *in situ* paradigm proposes to start processing data as soon as made available by the simulation, while still residing in the memory of the compute node. *In situ* processing includes data compression, indexing, computation of various types of descriptors (1D, 2D, images, etc.). The amount of data to save to disk is reduced, hence reducing the pressure on the file system when writing the results, as well as when reading them back for the postmortem analysis. Results can also be visualized on-line, enabling advanced execution monitoring.

Per se, reducing data output to limit I/O related performance drops or keep the output data size manageable is not new. Scientists have relied on solutions as simple as decreasing the output frequency. *In situ* processing proposes to move one step further, by providing a full fledged processing framework enabling scientists to more easily and thoroughly manage the available I/O budget. The first publication coining the *in situ* term in this context is very likely by Kwan Liu Ma et al. [MWYT07]. Solutions emerged from existing visualization libraries like VTK or Visit that added new readers to get data directly from a live simulation [FMT+11, WFM11], or from I/O libraries like ADIOS [LKS+08] augmented with processing capabilities when transiting data from the simulation to the file system [ZZE+13].

The most direct way to perform *in situ* analytics is to inline computations directly in the simulation code. This is the approach adopted in [YWG+10] as well as for the standard visualization tools like Paraview and Visit [FMT+11, WFM11] and its recent extensions [AWW+16, LAA+17]. In this case, *in situ* processing is executed in sequence with the simulation that is suspended meanwhile. We refer to this approach as *synchronous*.

To improve resource usage *asynchronous in situ* proposes to overlap the simulation and analysis executions. A direct approach consists in relying on the OS scheduler capabilities to allocate resources. The analytics run its own processes or threads concurrently with the ones of the simulation. The simulation only needs to give a copy of the relevant data to the local *in situ* analytics processes. The analytics can next proceed concurrently with the simulation. However, some works [ZYH+13, HMM14] show that relying on the OS scheduler does not prove efficient because the presence of analytics processes tends to disturb the simulation (context switch, cache trashing). GoldRush proposes to activate analysis executions only on long-enough sequential sections of the simulation. Tins adopts a task-based programming relying on a work-stealing scheduler for a fine grain interleaving and load balancing of tasks on the

compute nodes [DCR18].

To reduce these interferences, several works propose to rely on space sharing where one or several cores per node, called *helper cores*, are dedicated to analytics. Damaris [DAC+12], FlowVR [DR14], Functional Partitioning [LVB+10], GePSeA [SBF09], Active Buffer [MLW06] or SMART [WABJ15] adopt this solution. Tins [DCR18] introduced dynamic helper cores, dedicating cores to analysis only when analysis tasks are ready to be run. Even if the *in situ* processing simply consists in saving data to disks, this approach can be more efficient than to rely on standard I/O libraries like MPI I/O [DAC+12]. The simulation runs on less cores, but, because it is usually not 100% efficient, the performance is decreased by less than the ratio of confiscated cores. Still, helper isolation is limited to the compute core and the 2 first cache levels. Memory and usually the L3 cache are shared between cores. *In situ* tasks, by trashing the L3 cache or using a significant amount of the node memory can affect the simulation performance. Communication that can perform *in situ* tasks can also load the network interface and slow down the simulation communications. To better schedule the communication load, DataStager [AWE+09] proposes a mechanism to trigger *in situ* related traffic outside of the simulation communication phases.

For a better isolation of the simulation and *in situ* processes, one solution consists in offloading *in situ* tasks from the simulation to the costs of moving the data from the simulation nodes to the staging nodes. HDF5/DMS [BSO+11] uses the HDF5 and PnetCDF I/O library calls to expose a virtual file to staging nodes. GLEAN [TRB+08, VHP11] is another example of a simulation/visualization coupling tool initiated by making HDF5 and PnetCDF I/O library calls. DataSpaces [DPK12] stores the simulation data on staging nodes with a spatially coherent layout and acts as a server to client applications. Padawan [CML18] proposes a Python based staging solution. Several systems use staging nodes to expose the simulation data to other scientific workflows [DSS+05, LAB+06].

A few frameworks support both *in situ* and *in transit* processing. JITStaging [AEW+11] and PreData [ZAD+10] propose to extract data from the simulation, apply a first *in situ* treatment with simple stateless codes, then transfer the data to staging nodes. Bennett et al. [BAB+12] solution is build on top of DataSpaces and Dart [DPK08] to perform *in situ* and *in transit* visualization and analytics. FlexIO [ZZE+13] brings to ADIOS *in situ* and *in transit* processing capabilities. FlexIO uses shared memory segments to handle data to asynchronous node-local *in situ* processes and RDMA transport methods for inter-node transfers, in particular to staging nodes. Specific stateless codelets can be dynamically moved on different cores during the simulation. For NxM like data re-distributions, FlexIO relies on centralized coordinators that gather information about data and process distribution, compute the communication pattern and send back the necessary instructions to each process involved. This handshaking process can be totally or partly

bypassed if the data distribution does not need to be recomputed in between consecutive steps. Zhang et al. [ZDP+12] added a shared memory space to a Dart server to support both simulation code coupling and *in situ*/*in transit* scenarios. The user describes groups of parallel codes called bundles and creates a workflow between these bundles. Based on MPI, the framework requires that all bundles are integrated in the same MPI application, which can require significant coding efforts. Similarly, Damaris [DSP+13, DAC+12] proposes to embed *in situ* tasks in the MPI context of the simulation. At launch time, MPI processes define the type of task they execute (simulation or *in situ*) depending on their mapping on the target machine. Then, cores or nodes can be dedicated to *in situ* or *in transit* tasks. Bredala [DP16] enables automatic global data redistribution between the *in situ* and *in transit* nodes as well as a contract to extract data from the simulation depending on the analysis demands [MDRP17]. Having a single MPI application is interesting for some supercomputers OS that do not support running more than one application per node. FlowVR [DR14] relies on a dataflow model where components can be mapped *in situ* or *in transit* without the constraint of having all components running in the same MPI context.

Most of these approaches are MPI+X based. New programming models are also developed as alternatives to message passing. StarPU [ATNW11], PaRSEC [HHBD17], Legion [BTSA12] and HPX [KHAL+14] propose task-based runtime systems for distributed heterogeneous architectures. The program defines a directed acyclic graph where vertices are tasks and edges data dependencies between tasks. The runtime is in charge of mapping tasks to resources, and triggering task execution and the necessary data movements when data dependencies are resolved. Early experiments have been reported using Legion for *in situ* analytics [PBH+16, HSP+17]. They show that Legion runtime is able to overlap analytics and simulation tasks, but globally the performance is not yet competitive with MPI approaches.

A limited number of works have addressed *in situ* application modeling to define algorithms, study scheduling policies and resource partitioning. A paper [LHK+16] proposes a statistical performance model, based on algorithmic complexity to predict the run-time cost of a set of representative parallel rendering algorithms. These are commonly used for *in situ* rendering, but does not take into account the interactions between the simulation and the analytics components. Another group of works based on DataSpace focus on optimizing in transit data storage. Some studies [SJR+15] focus on data placement. Stacker [SDD+18] relies on machine learning to optimize the placement of data on the memory hierarchy, taking into account per node persistent storage capabilities like NVRAM or Burst Buffers. Zheng et al. [ZZE+13] also propose several heuristics to compute process to core mappings and optimize the use of helper cores and staging nodes. Malakar et al. [MVM+15, MVK+16] considered *in situ* analysis as a numerical optimization problem to compute an optimal frequency of an-

alytics subject to resource constraints such as I/O bandwidth and available memory. However, their work is limited to sequential simulation and analysis.

3 Model

In this section, we present the model used for scheduling a simulation and the associated analytics on a HPC platform. The model presented in this section is based on the assumption that the applications generate lots of data that can be analyzed on-line.

3.1 Architecture

The platform is composed of C_n identical unit-speed nodes. Each node is composed of c cores (also called processors throughout this work) and a shared memory of size M_n between all the cores of a node.

We model communications as an extra cost. Intra-node communications are considered as cost-free due to the shared memory space between processors. We assume that a processor can access the data on its node memory without extra cost. However, inter-node communications generate an overhead that is modeled as follows.

We define a communication cost $V \mapsto T_{\text{com}}(V)$: assume a volume of data V located on the memory of node N_i, then it takes $T_{\text{com}}(V)$ units of time to transfer it on the memory of node N_j.

In this work we use the linear bandwidth model [WWP09] to model the communication cost[1]:

$$T_{\text{com}}(V) = \frac{V}{b}$$

where b is the available bandwidth for the transfer. We assume that total bandwidth of the system is equally divided between each node, as is the total memory for each node.

Note that other classical communication models take into consideration a latency that fades-out for large messages. We consider here significant data movements, hence this latency has a negligible effect in our case.

3.2 Applications

This work focuses on iterative HPC applications that consist pf two different phases: the simulation phase, a compute-intensive phase that generates large amounts of data, and the analysis phase, a data-intensive phase that consists in transforming the data generated in the previous phase. Our model support both *in situ* and *in transit* analysis. Part of the analysis phase can be executed *in situ* on the nodes used for the simulation either in a synchronous or asynchronous mode. The other part runs *in transit* on dedicated nodes, generally called staging nodes.

[1] Although, most of this work is agnostic of the bandwidth model used

We consider that all tasks, being simulation or analysis tasks, are moldable: the scheduler can decide prior to execution how many nodes and cores are assigned to each task. Once the application starts, this number is fixed for the full computation duration.

The work of a task is defined as the execution time of the task times the number of cores used. We consider here tasks where the amount of work is constant with the increase in pluralization (*perfectly/embarrassingly parallel* model [HS11]). Put differently, parallelizing the application implies no overhead. Other models such as Amdahl law [Amd67] can be used. For readability we focus the next sections on the perfectly parallel model, but most of the results are valid with other work models, and when needed we discuss in this paper the impact of such models.

Each task/function (simulation or analysis) is defined by two parameters:

- a reference execution time given as a running time on one core; and
- a memory peak representing the maximum memory consumption of the task during its execution.

For notation, we introduce $t^{\rho}_{\mu}(c)$ as the makespan of task μ (either simulation or analysis function) at iteration ρ on c core(s).

3.2.1 Simulation Phase:

We consider the simulation as a job iterated Π times. In the following, let S_ρ ($\rho \in \{1, \cdots, \Pi\}$) be the ρ^{th} simulation iteration.

Each iteration S_ρ is defined by its data input V_ρ, its execution time $t^{\rho}_{\text{sim}}(1)$ and its memory peak P_ρ.

Assuming there is enough memory available to perform the iteration S_ρ, let $t^{\rho}_{\text{sim}}(c)$ be the execution time on c cores to perform the simulation,

$$t^{\rho}_{\text{sim}}(c) = \frac{t^{\rho}_{\text{sim}}(1)}{c}$$

We also work under the assumption that the memory peak does not depend on the number of cores working on the iteration. Hence for iteration S_ρ running on c cores, the average peak memory per core is $\frac{P_\rho}{c}$.

We consider that the simulation data are evenly distributed amongst the nodes assigned to the simulation, and thus that the simulation outputs are also evenly distributed. We consider that the simulation does not perform I/Os directly. Simulation outputs are all handed to analysis tasks that are in charge of data processing and eventually to perform I/O to save the necessary data to disk.

3.2.2 Analysis Phase:

An analysis phase runs after each simulation iteration S_ρ. We denote by $\mathcal{A}^\rho$ the set of analysis tasks to be performed on the output data of iteration S_ρ:

$$\mathcal{A}^\rho = \{A_1^\rho, \cdots, A_{K^\rho}^\rho\}$$

Any of the tasks of $\mathcal{A}$ can be executed either *in situ* or *in transit*. We denote by $\mathcal{A}^{IS}$ and $\mathcal{A}^{IT}$ the partition of the analytics tasks according to their execution mode:

$$\mathcal{A}^{IS} \dot{\cup} \mathcal{A}^{IT} = \mathcal{A}$$

For this work, we assume that all analysis tasks run at each simulation iteration ρ are identical:

$$\mathcal{A}^\rho = \mathcal{A}$$

Analytics tasks can include about anything, like computation of high level descriptors [DCR18], compressing data [TDCC17], verifying the integrity of the data to detect a silent error [ABH+13], or checkpointing for reliability [ARVZ14].

Analytics tasks can either be executed synchronously, i.e. without overlap with the simulation or asynchronously. We detail both scenarios in coming paragraphs.

For now, we recall that $t_i(c)$ denotes the processing time of function i on c cores.

***in situ* Analysis:** The *in situ* analysis tasks directly access the local simulation output data on the node where they run. As all tasks run in parallel on each simulation node, the total makespan of *in situ* analysis functions is the sum of the makespan of each task.

The *in situ* tasks need memory space allocated in the simulation nodes. We ensure that this does not impair the simulation execution by enforcing that the sum of the peak memory of both the simulation and analytic tasks running on the simulation node do not overflow the total memory of each simulation node.

If we perform synchronous analytics, the simulation is paused to perform *in situ* analytics on simulation resources. It means that simulation and analysis use all the cores of their assigned nodes. For asynchronous analytics, we used so called *helper cores* to perform the *in situ* analysis. In this case, analysis and simulation overlaps. Because of helper cores, the simulation is slowed down and a trade-off between analysis benefits and simulation performance loss has to be addressed.

I/O are performed to transfer the output of analytics to the PFS. As it is assumed to have negligible cost, the cost is included into the function makespan.

***in transit* Analysis:** The input to the $\mathcal{A}^{IT}$ functions are not available on the *in transit* nodes. They need to be transferred from the simulation nodes.

We model the *in transit* cost in terms of time only. We assume that the cost is the time to send all the input of functions $\mathcal{A}^{IT}$ (i.e all associated memory peaks) from simulation nodes to *in transit* nodes, associated to the cost for running the functions. The output of analysis functions are stored on PFS, which is assumed to induce a negligible cost that is not subject to interference. Thus, we consider that this cost is included in the makespan of the functions. We also consider that it is the *in transit* nodes that has to face the transfer overhead while the *in situ* nodes on which data are located can continue working.

In both asynchronous/synchronous scenarios, *in transit* analytics is performed on a dedicated set of nodes.

3.3 Application Pipeline

We end the presentation of the model by the application pipeline.

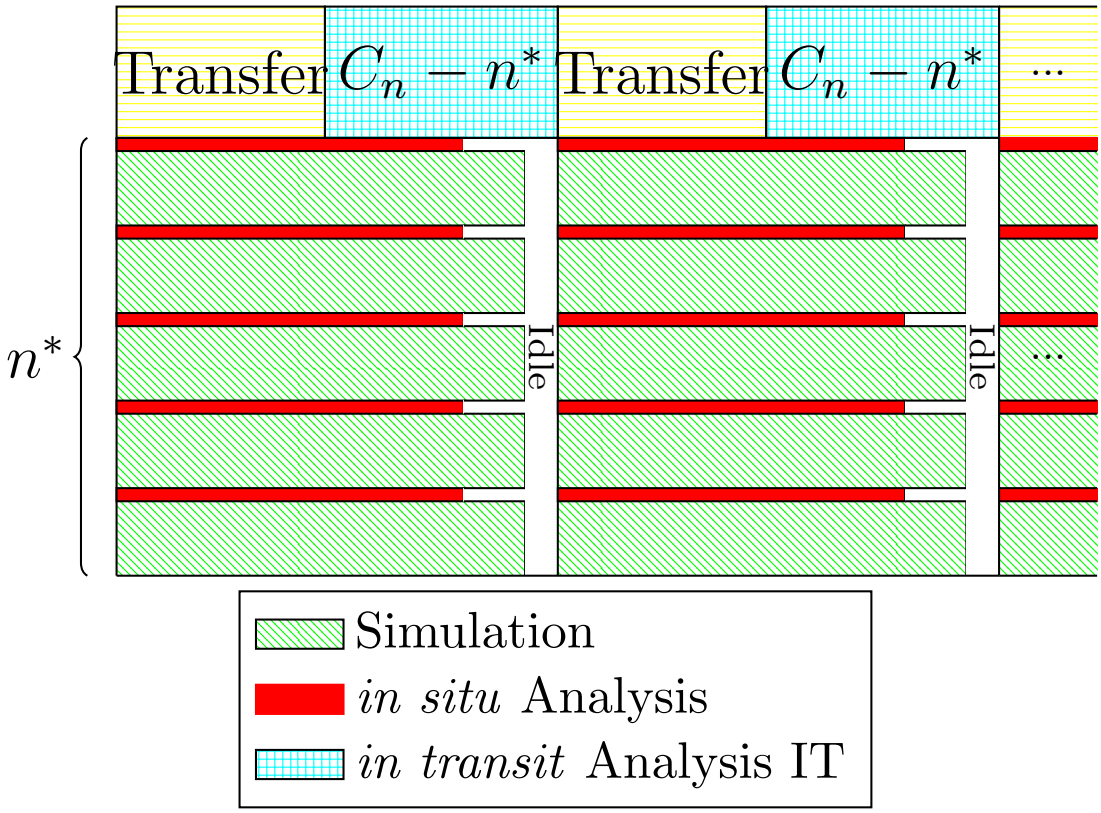

Figure 1: Illustration of Application Workflow with Asynchronous Analytics.

Figure 1 presents the application pipeline for asynchronous analytics. In this model, the cores dedicated to *in situ* analysis cannot be used for the simulation. They are called helper cores. For each simulation iteration, the set $\mathcal{A}^{IS}[\rho - 1]$ is executed concurrently to the ρ^{th} simulation round on n^* *in situ* nodes. Note that they do not need necessarily to be executed concurrently, but this is one of the strategies that minimizes the execution time since the helper cores cannot be used for anything else that analysis.

There are c^* helper cores over the c cores that are dedicated to perform *in situ* $\mathcal{A}^{IS}$ functions. Thus, $c - c^*$ cores are dedicated to the simulation iterations. The helper cores can access simulation data locally, without requiring to transfer the input of the analysis functions. However, it reduces the number of cores dedicated to simulation, and its performance. The set of *in transit* functions is performed on $C_n - n^*$ nodes, using all available cores. However, the transfer of function inputs induces a time overhead that has to be balanced with the makespan of simulation and *in situ* analysis. Finally, the application makespan is computed as the maximum time between simulation, *in situ* and *in transit* makespan. Some idle time on resources appears when one of these makespans is longer than another.

Figure 2 presents the application pipeline for synchronous analytics. The main difference with the previous scenario is that the *in situ* analytics is processed in sequence with the simulation, that is paused during that time. The *in transit* analytics follows the same rules as before. As for asynchronous case, some idle time may appear, for example if the *in transit* analytics has a larger makespan than the simulation followed by *in situ* analysis. In synchronous case, the application makespan is the maximum between the simulation summed with the *in situ* analytics and the *in transit* makespan.

In the rest of the paper, we perform the analysis using the asynchronous scenario. However, in Section 6, we discuss the performance of both asynchronous and synchronous analytics.

3.4 Resource Allocation Optimization Problems

In this section, we define properly the resource partitioning problems using our application and platform models. Firstly, we define a general problem. Recall that we process all analysis functions at each iteration of simulation. We now propose Problem 1, called 1-ARP (1-Application Resource Partitioning) problem:

Problem 1 (1-Application Resource Partitioning (1-ARP))**.** *Given $\mathcal{A}$ the set of analysis tasks, $\mathcal{S}$ the set of simulation tasks, can we determine n^* the number of simulation nodes, c^* the number of helper cores and a scheduling of the N calls to the set $\mathcal{A}$ such that the total makespan of the application is minimal?*

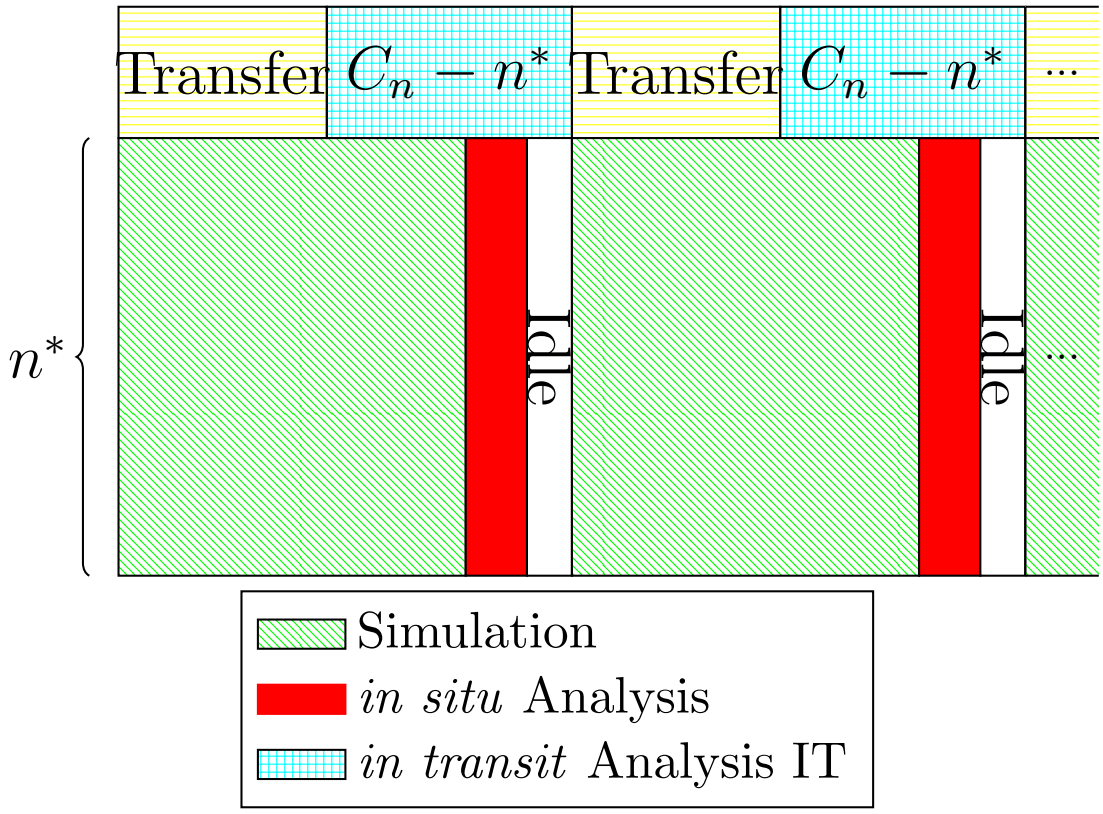

Figure 2: Illustration of Application Workflow with Synchronous Analytics.

This is the general problem we will solve in order to get an efficient distribution of application workload over an optimal distribution of computational resources.

Before solving 1-ARP, we study a refinement of it when the analysis sets are given. This is express as Problem 2.

Problem 2 (Resource Partitioning for Analysis Sets (RePAS)). *Given $\mathcal{A}^{IS}$, $\mathcal{A}^{IT}$ and $\mathcal{S}$, can we determine n^*, c^* and a scheduling of the N calls to the set $\mathcal{A}$ such that the total makespan of the application is minimal?*

The simpler problem RePAS provides a solution for 1-ARP when the scheduling of the analysis functions (either in situ or in transit) is given. We show in the next section how to compute a solution to RePAS. We then use this solution in Section 5 to derive solutions to 1-ARP.

4 Resource Partitioning for Analysis Sets

In this section, we study the RePAS problem. We first compute an optimal floating solution for the number of *in situ* nodes n^* and the number of helper cores c^*. We will then discuss an integer solution for this problem.

4.1 Reformulation of RePAS

We have seen in Section 3.3 that in the asynchronous case, the analysis from the iteration $i-1$ are performed at the same time as the simulation of iteration i. Hence the total execution time consists of the sum of: (i) the first iteration of the simulation; (ii) $\Pi-1$ times the maximum time taken, either by the simulation, or by the analysis times; and (iii) the time for the final analysis.

Let us now denote by:

- $T_S(n^*, c^*)$, the time to execute one iteration of the simulation on n^* nodes, given that for each node, c^* cores are dedicated for in situ analysis;
- $T_A^{IS}(\mathcal{A}^{IS}, n^*, c^*)$ the time to perform the analysis $\mathcal{A}^{IS}$ on n^* nodes, using c^* cores per node; and
- $T_A^{IT}(\mathcal{A}^{IT}, n^*)$ the time to perform the analysis $\mathcal{A}^{IT}$ given that the simulation is using n^* nodes.

Then the execution time is:

$$T_S + (\Pi - 1)\max\left(T_S, T_A^{IS}, T_A^{IT}\right) + \max\left(T_A^{IS}, T_A^{IT}\right)$$

Assuming that Π is large enough, then

$$T_S + \max\left(T_A^{IS}, T_A^{IT}\right) << (\Pi - 1)\max\left(T_S, T_A^{IS}, T_A^{IT}\right)$$

and we can focus on the following optimization problem:

Problem 3. *Given* $(\mathcal{A}^{IS}, \mathcal{A}^{IT})$*, find* $n^* \leq C_n$*, and* $c^* \leq c$ *that minimize*

$$\max\left(T_S(n^*, c^*), T_A^{IS}(\mathcal{A}^{IS}, n^*, c^*), T_A^{IT}(\mathcal{A}^{IT}, n^*)\right)$$

In the following we show how to compute a solution to this problem, that is how to find n^* and c^*.

We first write formally the different execution times.

Proposition 1 (Execution time of the different phases)**.** *Let X be the sum of the single core execution time of each $A_i \in \mathcal{A}^{IS}$ (i.e. $X = \sum_{A_i \in \mathcal{A}^{IS}} t_i(1)$). Similarly, we have: $\sum_{A_i \in \mathcal{A}^{IT}} t_i(1) = W - X$ where W is the total time of analysis tasks. Let Mem^{IT} be the sum of the memory peak of each $A_i \in \mathcal{A}^{IT}$ and b the bandwidth per node of the platform.*

$$
\begin{aligned}
T_S(n^*,c^*) &= \frac{t_{sim}(1)}{n^* \cdot (c - c^*)} \\
T_A^{IS}(\mathcal{A}^{IS},n^*,c^*) &= \frac{X}{n^* c^*} \\
T_A^{IT}(\mathcal{A}^{IT},n^*) &= \left(\frac{W - X}{c(C_n - n^*)}\right) + T_{Com}\left(n^*, C_n - n^*, \mathcal{A}^{IT}\right) \\
&= \left(\frac{W - X}{c(C_n - n^*)}\right) + \frac{Mem^{IT}}{(C_n - n^*) \cdot b}
\end{aligned}
$$

These different costs come naturally from the fully parallel model, indeed it is the time to sequentially run all the tasks divided by the number of cores used.

4.2 Solution with rational number of cores and nodes

In this section, we compute a solution to Problem 3.

Theorem 1 (Optimal Number of Helper Cores)**.** *The optimal number of helper cores is given by the following:*

$$
c^* = \frac{(X \cdot c)}{t_{sim}(1) + X}
$$

Proof. To obtain this result, one can verify that in an optimal solution,

$$
T_S(n^*, c^*) = T_A^{IS}(\mathcal{A}^{IS}, n^*, c^*).
$$

Indeed, otherwise if one is faster than the other one, we can share ε cores from the fastest computation such that its execution time does not increase too much. This will reduce the slowest computation hence contradicting the optimality of the solution.

Hence we have:

$$
\begin{aligned}
\frac{t_{\text{sim}}(1)}{c - c^*} &= \frac{X}{c^*} \\
t_{\text{sim}}(1) \cdot c^* &= c \cdot X - c^* \cdot X \\
c^* &= \frac{X \cdot c}{t_{\text{sim}}(1) + X}
\end{aligned}
$$

□

Using the value of c^* from Theorem 1, we now compute n^*.

Theorem 2 (Optimal Number of in situ Nodes)**.** *The optimal number of* in situ *nodes is given by:*

$$n^* = \frac{X \cdot C_n}{c^* \cdot \left(\frac{W-X}{c} + \frac{Mem^{IT}}{b}\right) + X}$$

Proof. The result is obtained similarly to Theorem 1. First according to Theorem 1, we know that in the optimal solution the time for in situ analysis is equal to the simulation time. In addition, we use the formula from Problem 3 and verify that:

$$T_A^{IS}(\mathcal{A}^{IS}, n^*, c^*) = T_A^{IT}(\mathcal{A}^{IT}, n^*),$$

where c^* is the value obtained in Equation (1) (and is a function of n^*). Similarly to the previous Theorem, one can verify that if one of the analysis is faster than the other one, we can share ε nodes from the fastest analysis such that its execution time does not increase too much. These nodes are then allocated to the other analysis which will reduce its execution time hence contradicting the optimality of the solution.

As we did before, we will solve the following equation in order to obtain the value of n^*:

$$\begin{aligned}\frac{X}{n^*c^*} &= \left(\frac{W-X}{c(C_n-n^*)}\right) + T_{Com}\left(n^*, C_n - n^*, \mathcal{A}^{IT}\right) \\ &= \left(\frac{W-X}{c(C_n-n^*)}\right) + \frac{Mem^{IT}}{(C_n-n^*)\cdot b}\end{aligned}$$

By rewritting,

$$\frac{X}{c^*} = \frac{n^*}{(C_n-n^*)} \cdot \left(\frac{W-X}{c} + \frac{Mem^{IT}}{b}\right)$$

Then,

$$n^* \cdot c^* \cdot \left(\frac{W-X}{c} + \frac{Mem^{IT}}{b}\right) = X \cdot (C_n - n^*)$$

This results in

$$n^* \cdot c^* \cdot \frac{W-X}{c} + n^* \cdot c^* \cdot \frac{Mem^{IT}}{b} + X \cdot n^* = X \cdot C_n$$

Finally,

$$n^* = \frac{X \cdot C_n}{c^* \cdot \left(\frac{W-X}{c} + \frac{Mem^{IT}}{b}\right) + X}$$

□

This result is derived for a linear bandwidth model but similar derivations can be performed with other communication models.

4.3 Integer Solution for RePAS Problem

In Section 4.2, we described a solution to compute the number of *in situ* nodes n^* and the number of helper cores c^*. However, those solutions return a float number which is not suitable for us to describe a number of system physical resource.

To solve this issue, we round the result to the closest higher integer. Recall that in the proof of Lemma 1, the makespan of *in situ* analysis is computed to be at most equal to the simulation one, to avoid performance loss. Thus, we choose to round c^* and n^* to highest value to ensure that the highest makespan will be the simulation (in other words, analysis does not penalize simulation in the current setup).

This can lead to idle time for *in situ* resources. This will be the target of a future work to evaluate the best policy for c^* and n^* rounding.

In the following, we consider $\Pi = 1$ without loss of generality.

5 Scheduling Strategies for High-Throughput Applications

With the solution of Problem 3, we now want to solve the general 1-Application Resource Partitioning problem. Indeed, we are now able to determine a resource partitioning given a division of the analysis tasks between *in situ* and *in transit* processing. However, all the possible schedules are not always suitable, due to the limited amount of memory of the system. As we discussed in Sections 3.2.2, let us define procedure *VIABILITY* that, for a given system, verifies that the *in situ* nodes have enough memory for simulation and the memory peaks associated to the scheduled *in situ* analysis. As we previously stated, the set of *in transit* analysis do not require memory to be performed (cf. Section 3.2.2). However, they generate a communication time for sending their input to the *in transit* nodes. We now propose different scheduling heuristics for simulation and analysis tasks.

5.1 "One-by-One" Greedy Algorithms

A natural polynomial strategy for scheduling algorithm is to greedily move analysis functions from $\mathcal{A}^{IT}$ to $\mathcal{A}^{IS}$, and to compute the optimal allocation of nodes and cores. Note that because of memory constraints, not all analysis can fit *in situ*, which is why we consider that they all start as *in transit* analysis.

Such an algorithm is described by Algorithm 1. The idea is to sort the analysis functions following a given metric (priority order). We initialize the algorithm with $\mathcal{A}^{IT} = \mathcal{A}$ and $\mathcal{A}^{IS} = \emptyset$. Then we greedily move each analysis according to the priority order one by one from $\mathcal{A}^{IT}$ to $\mathcal{A}^{IS}$. When an analysis is moved, we determine the minimum number of nodes needed so that the required memory for simulation plus the memory reserved for *in situ* analysis is not greater than the memory available. If the number of nodes is greater than C_n, then we leave this application in $\mathcal{A}^{IT}$. Otherwise we compute the optimal allocation of nodes and cores following Lemma 3. Procedure 2, describes by Algorithm 2, ensures the respect of memory constraints.

In this work we try the following priority functions:

- Increasing/Decreasing Time: we sort the analysis by their increasing/decreasing execution time on a single core.
- Increasing/Decreasing Peak: we sort the analysis by their increasing/decreasing memory peak.
- Random: we compute a uniformly random priority order.

Algorithm 1: *Generic Greedy Algorithm*

Require: Set of analysis Ana, total number of nodes C_n, total number of cores c memory per node m, bandwidth per node b, simulation time $\mathcal{S}$, memory consumption of simulation m^S, a metric *metric* to sort the analysis

Ensure: A resource partitioning n^*, c^* and a scheduling of tasks Ana^{IS}, Ana^{IT}

1: $Ana^{IS} \leftarrow []$
2: $Ana^{IT} \leftarrow Ana$
3: $Analysis \leftarrow metric(Ana)$
4: $(n^*, c^*) \leftarrow PTNG\left(Ana^{IS}, Ana^{IT}, C_n, c, b\right)$
5: exec_time $\leftarrow SCHED\left(C_n, n^*, c, c^*, \mathcal{S}, Ana^{IS}, Ana^{IT}, b\right)$
6: **for** $a_i \in Analysis$ **do**
7: $\quad Ana^{IS} \leftarrow Ana^{IS} \cup a_i$
8: $\quad Ana^{IT} \leftarrow Ana^{IT} \setminus a_i$
9: $\quad (n_1, c_1) \leftarrow PTNG\left(Ana^{IS}, Ana^{IT}, C_n, c, b\right)$
10: $\quad$ **if** $VIABILITY\left(Ana^{IS}, n_1, m, m^S\right)$ **then**
11: $\quad\quad$ e $\leftarrow SCHED\left(C_n, n_1, c, c_1, \mathcal{S}, Ana^{IS}, Ana^{IT}, b\right)$
12: $\quad\quad$ **if** e $<$ exec_time **then**
13: $\quad\quad\quad n^* \leftarrow n_1$
14: $\quad\quad\quad c^* \leftarrow c_1$
15: $\quad\quad$ **else**
16: $\quad\quad\quad Ana^{IS} \leftarrow Ana^{IS} \setminus a_i$
17: $\quad\quad\quad Ana^{IT} \leftarrow Ana^{IS} \cup a_i$
18: $\quad\quad$ **end if**
19: $\quad$ **else**
20: $\quad\quad Ana^{IS} \leftarrow Ana^{IS} \setminus a_i$
21: $\quad\quad Ana^{IT} \leftarrow Ana^{IS} \cup a_i$
22: $\quad$ **end if**
23: **end for**
24: **return** $\left(Ana^{IS}, Ana^{IT}, n^*, c^*\right)$

Decreasing Time is well known to be efficient for scheduling applications with an objective of minimizing the execution time. Increasing Time is famously known to be efficient with respect to the objective of improving fairness. Random is here as a witness: anything worse than random can be considered as a poor order.

Algorithm 2: *Viability of a resource partitioning for a set of* in situ *analysis*

Require: Set of *in situ* analysis Ana^{IS}, number of *in situ* nodes n^*, memory per node m, memory consumption of simulation m^S

Ensure: $True$ if the *in situ* input system would be viable, $False$ otherwise

```
1:  result = False
2:  IS_memory ← (m × n*) − m^S
3:  if (IS_memory ≥ 0) then
4:      counter ← 0
5:      for ana ∈ Ana^IS do
6:          counter ← counter + (memory_peak (ana))
7:      end for
8:      if counter ≤ IS_memory then
9:          result ← True
10:     end if
11: end if
12: return result
```

5.2 Optimal Scheduling Algorithm

To compare the previous algorithms, we will use the optimal scheduling algorithm that tests all the possible *in transit*/*in situ* configurations and keep the one that generates the lowest execution time. This optimal algorithm is described by Algorithm 3.

5.3 Complexity Analysis

The greedy algorithms have a linear complexity on the number of analysis tasks $\mathcal{O}(K \log(K))$. *VIABILITY* procedures have a complexity in $\mathcal{O}(K)$ and *PTNG* in $\mathcal{O}(1)$.

The optimal algorithm has a complexity exponential on the number of analysis functions $\mathcal{O}(2^K)$. Indeed, it has to generate all possible subsets of K functions. As usually there is a small number of analysis functions, this exponential factor remains limited (cf. Table 1 in [DCR18] for an example).

Algorithm 3: *Optimal Algorithm*

Require: Set of analysis Ana, total number of nodes C_n, total number of cores c memory per node m, bandwidth per node b, simulation time $\mathcal{S}$, memory consumption of simulation m^S

Ensure: A resource partitioning n^*, c^* and a scheduling of tasks Ana^{IS}, Ana^{IT}

$Ana^{IS} \leftarrow []$
$Ana^{IT} \leftarrow Ana$
subsets $\leftarrow$ generate_subsets (Ana)
exec_time $\leftarrow SCHED\left(C_n, n^*, c, c^*, \mathcal{S}, Ana^{IS}, Ana^{IT}, b\right)$
$(n^*, c^*) \leftarrow PTNG\left(Ana^{IS}, Ana^{IT}, C_n, c, b\right)$
for set in subsets **do**
  $IS \leftarrow$ set
  $IT \leftarrow$ ana $\setminus$ set
  $(n, HC) \leftarrow PTNG\left(IS, IT, C_n, c, b\right)$
  if $VIABILITY\left(IS, n, m, m^S\right)$ **then**
    e $\leftarrow SCHED\left(C_n, n, c, HC, \mathcal{S}, Ana^{IS}, Ana^{IT}, b\right)$
    if e $<$ exec_time **then**
      $n^* \leftarrow$ n
      $c^* \leftarrow$ c
      $Ana^{IS} \leftarrow IS$
      $Ana^{IT} \leftarrow IT$
    end if
  end if
end for
return $\left(Ana^{IS}, Ana^{IT}, n^*, c^*\right)$

6 Evaluation

In this section, we present an evaluation of our model and scheduling strategies.

6.1 Simulation and Setup

To evaluate the different strategies, we designed a simulator to study the performance of the different scheduling algorithms coupled with our resource partitioning solution. Each scheduling algorithm of Section 5 is implemented and returns an execution time for a given simulation function, set of analysis and platform. The platform is given as a number of nodes, cores, a total amount of memory M and a bandwidth. We recall that every node has the same amount of memory and bandwidth.

Simulation is given as a processing time on one core and a memory consumption.

We fix as a constant the memory consumption of the simulation to be 1 and the total available memory of the system to be 1.33. This is justified by the fact that usually, simulation is a large task that requires most of the system resources.

The analysis functions are described by the same features as the simulation and are randomly generated regarding the following process. We fix a desired memory occupation M for the application (simulation and analysis) and a number of analysis functions K. Then, we randomly generate a set of K analysis for which the total amount of memory peaks sums to $M - 1$. The execution time on one core for each function are randomly picked between an upper bound and lower bound percentage of the simulation execution time. In the following, we fix K to 8.

The simulator is designed to support both asynchronous analysis scenario and synchronous (in this case the *in situ* analysis is part of the simulation). For the latter scenario, we use the same scheduling and model tools previously mentioned. The simulation memory peak and work are updated with the scheduled *in situ* analysis.

Given this simulator, we perform different evaluations of the model and algorithms by increasing the memory load of the application and studying the impact of the memory constraint on the analysis execution mode. We vary the memory load of the application from 1.05 to 2[2]. To ensure that the communication time does not interfere in the results, we tested different bandwidth per node values going from 10% of the total memory per unit time to 100%.

To ensure the reliability of results, we perform 130 samplings for each memory occupation and plot the average of the results for each algorithm. The number of nodes is 50 and number of cores per node is 8.

The simulator has been developed using SageMath[3]. The plots of this section are generated using R language. The code of the simulator and all details related to software dependencies[4], plot generation or installation instructions are freely accessible online[5].

6.2 Results in Asynchronous Scenario

In this section, we present the results of the above setup in asynchronous scenario.

Figure 3 presents the performance of algorithms where each node has a bandwidth equals to 10% of its memory per unit time. The first plot presents the execution

[2]When the memory load equals to 2, the memory consumption of analysis and simulation are equal. In reality, this extreme point is never reached.

[3]http://www.sagemath.org/

[4]The simulator has not been tested on other versions of SageMath than 8.1 but there should not have any problem as we use Sage as a runtime, we do not use its provided libraries.

[5]https://gitlab.inria.fr/vhonore/insitu_simulator

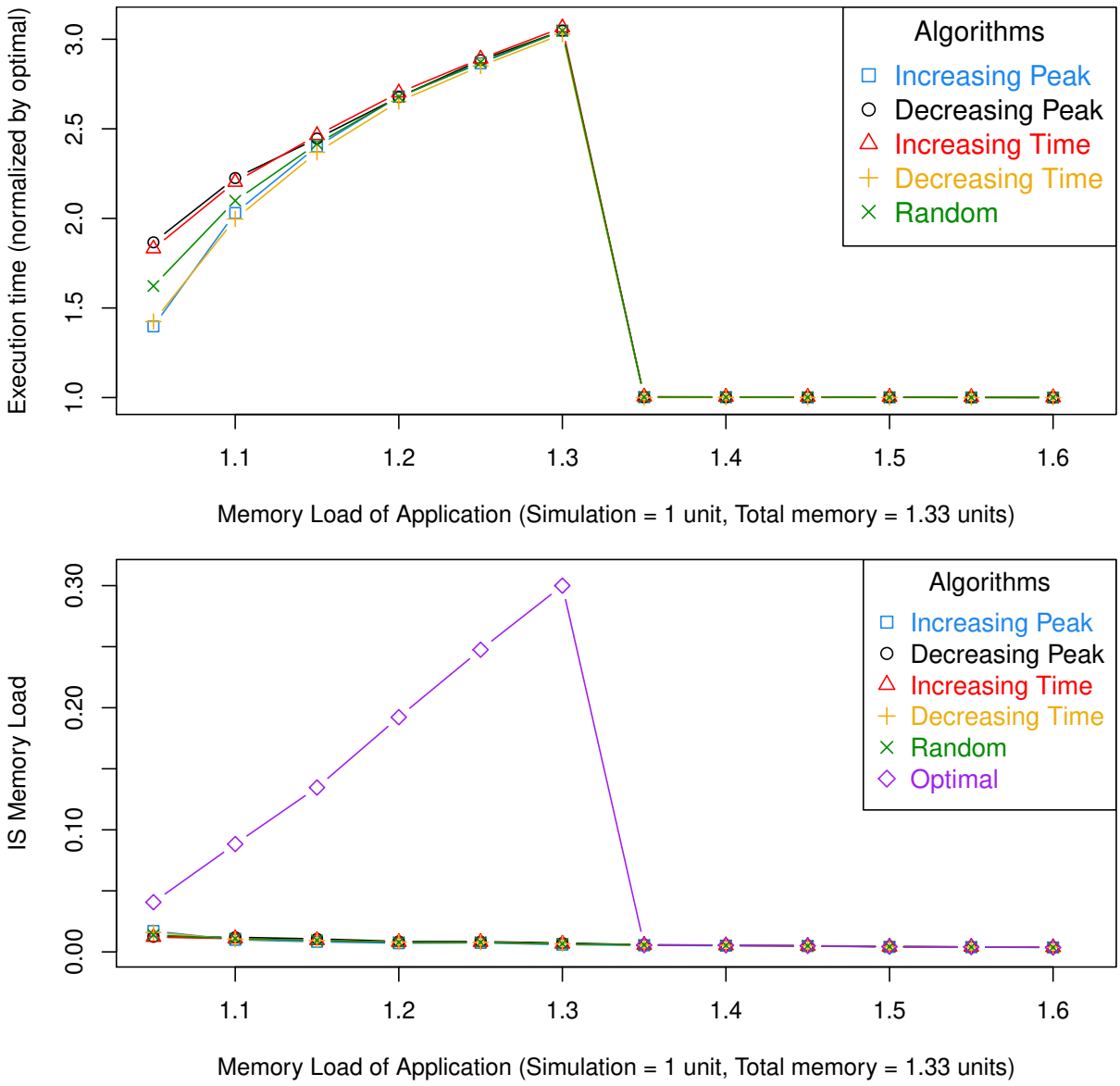

Figure 3: Simulation of Algorithm performance for asynchronous scenario with bandwidth is equal to 10% of the memory per node, per unit time.

time of the application with regards to memory occupation. The execution time are normalized with regard to the optimal Algorithm 3.

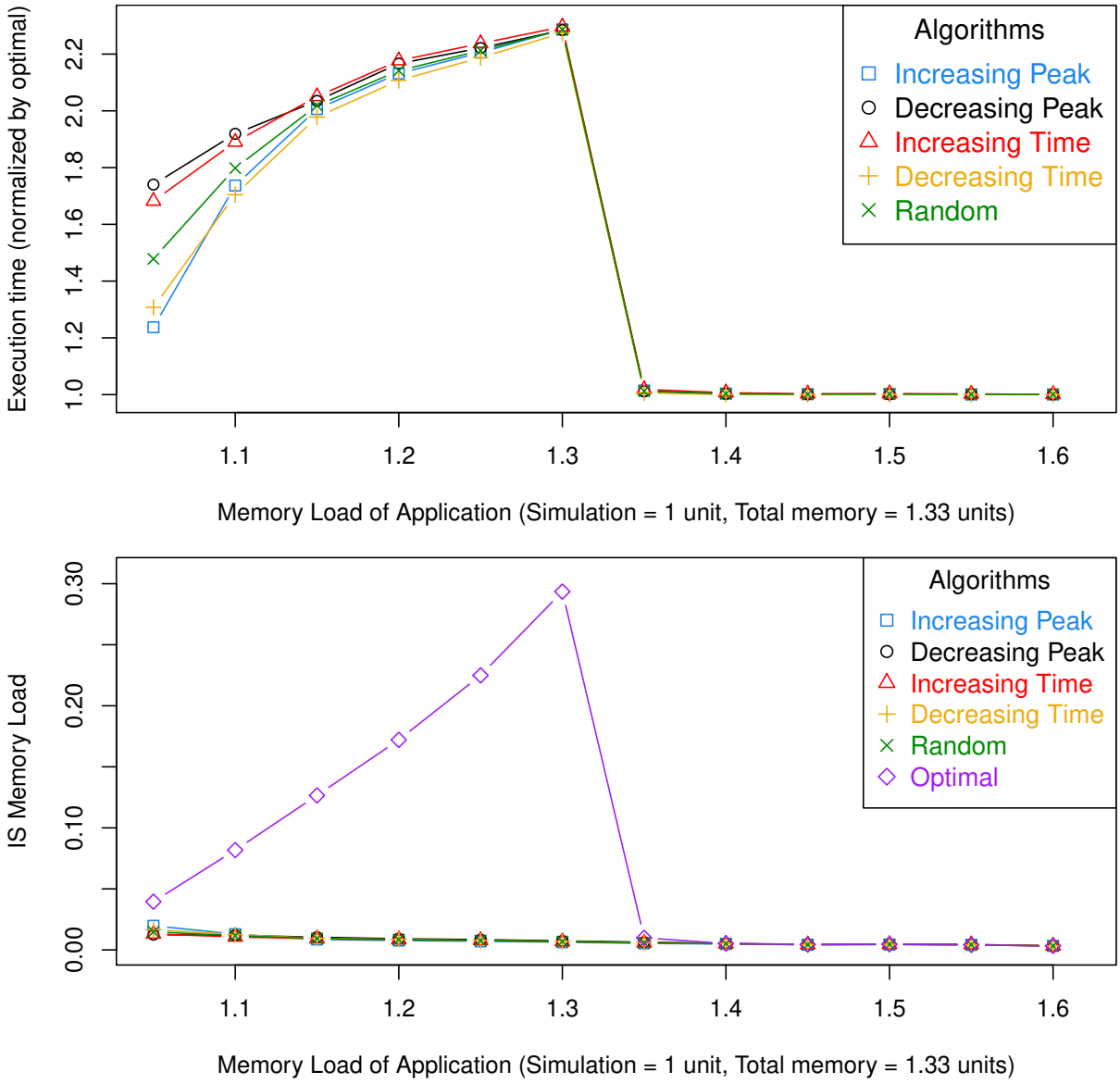

Figure 4: Simulation of Algorithm performance for asynchronous scenario with bandwidth equals to 25% of the memory per node, per unit time.

First of all, we note that the algorithms tend to converge to the optimal when either the constraints of memory are strong or weak. This is explained by the fact

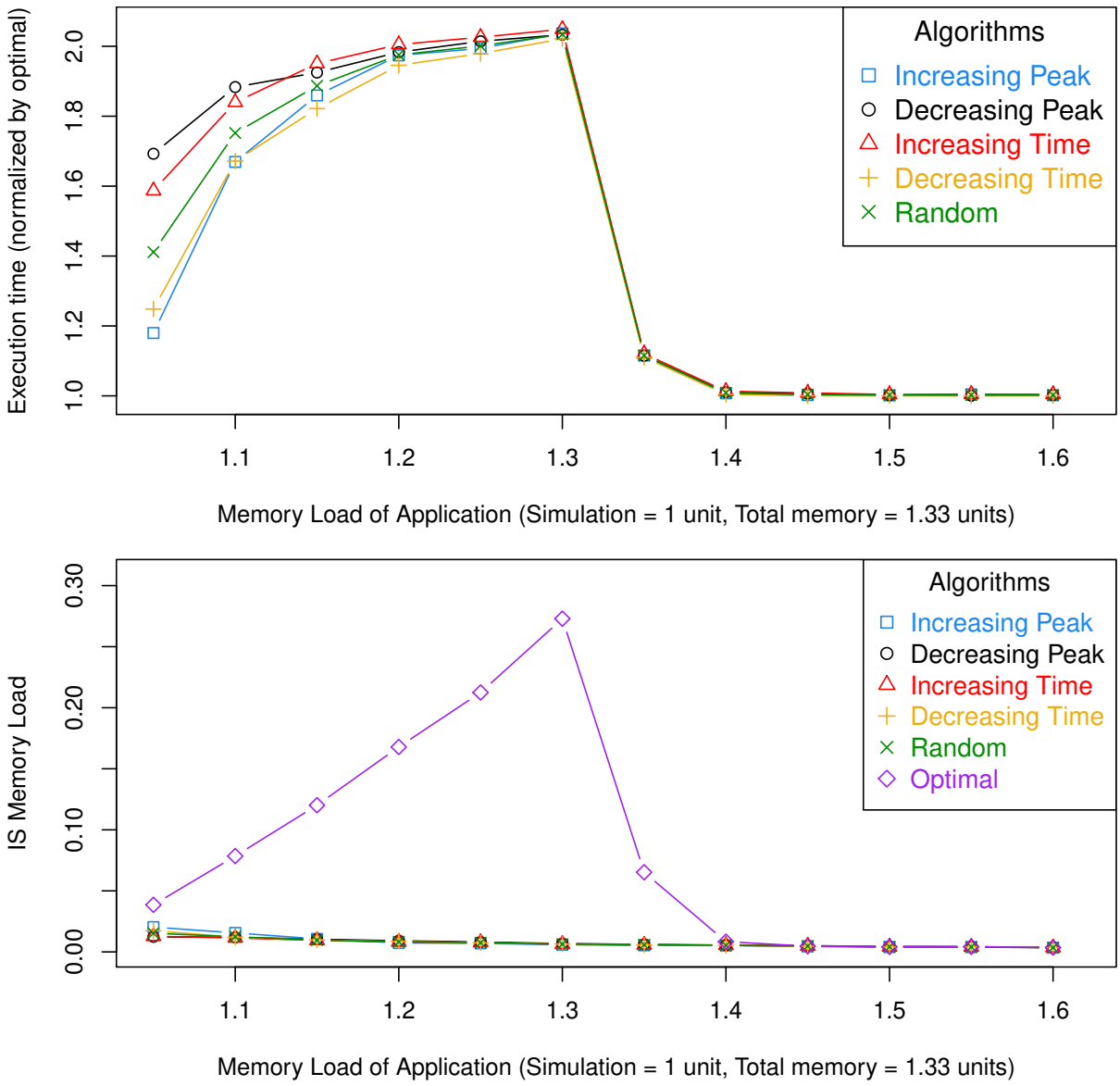

Figure 5: Simulation of Algorithm performance for asynchronous scenario with bandwidth is equal to 50% of the memory per node, per unit time.

that if the memory load of application overpasses by far the total memory of the platform, the only solution is to offload all the analysis *in transit* to maintain simulation

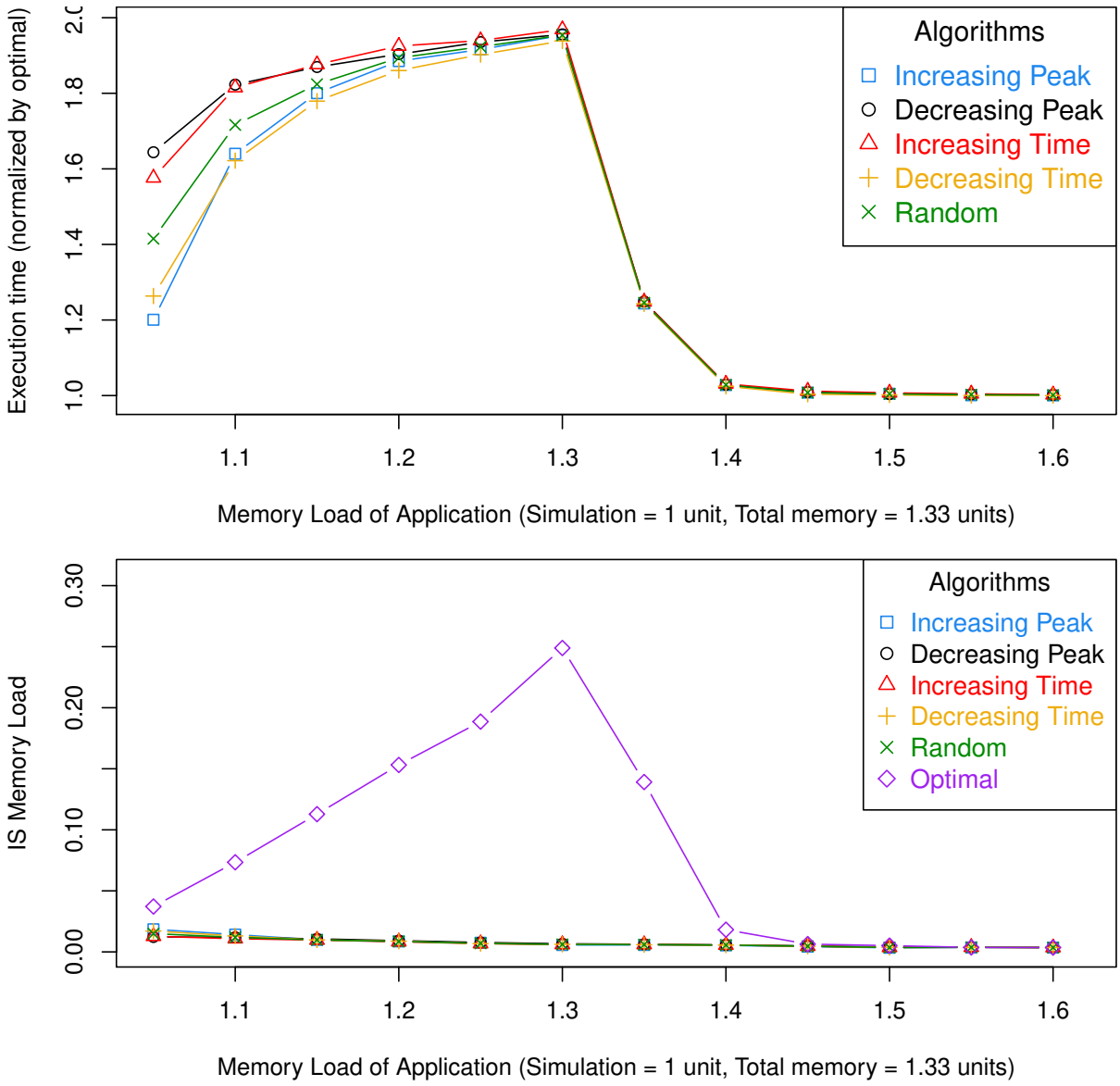

Figure 6: Simulation of Algorithm performance for asynchronous scenario with bandwidth is equal to 100% of the memory per node, per unit time.

performances (we assume that the simulation can fit on the platform by using all its resources). In the contrary, if the memory constraints are very low, the optimal

solution is to perform *in situ* analysis by sharing the resources between analysis and simulation. However, greedy heuristics are not able to find an appropriate *in situ* analytics schedule due to the fact that they consider analysis one by one. Figures 4 5 6 show that different bandwidth does not influence the scheduling policy.

From the performance analysis, we extract two algorithms that seems to perform better than others: Increasing Peak and Decreasing Time. They also induce an *in situ* memory occupation relatively close to the optimal. To understand part of their good performance, we have also plotted the *in situ* memory occupation for each algorithm with regards to the total memory occupation of the application. The observation that one can make is that the performance of algorithms seem correlated to the memory size occupied by *in situ* applications. It seems to make sense as one may expect that the additional cost incurred is the one due to data movement, hence one wants to minimize the amount of such movement.

However, an interesting conclusion is that our greedy algorithms are often performing badly with regards to this optimal. One of the reason is surely that those algorithms consider analysis one by one, and not by packs. It seems intuitive that scheduling a group of tasks on a given set of resources has more probability to induce better performance in an overall perspective than only one task. In the future, we must take into account batches of analysis rather than one by one in order to design efficient scheduling policies.

6.3 Criteria for Application Performance

From previous discussion on asynchronous scenario, one can deduce that an important feature for system performance is resource utilization. If some resources are reserved for *in situ* analysis, those resources must be sufficiently used in terms of memory and computation to improve system performance. Otherwise, those resources are better be left to simulation and the analysis offloaded into *in transit* processing.

It is not a trivial result as it seems. Indeed if an analysis is fast but uses lots of memory, it may mean that the helper cores are never used. One may expect that it would be better to use them. Note that this trade-off may explain the good performance of Decreasing Time. While Increasing Peak is better at using as much memory as possible for analysis, Decreasing Time might be better at using the helper cores as much as possible.

To validate this hypothesis, we plotted Figure 7. The x-axis shows the *in situ* memory load of considered algorithms (normalized by optimal algorithm). The more a point is on the right, the better the scheduling algorithm uses the *in situ* memory. The y-axis is dedicated to (normalize) execution time: the lower a point is, the better the associated algorithm performs. We included in this plot all points of Figure 3 to Figure 6. We shape each point by its algorithm and color it with a gradient

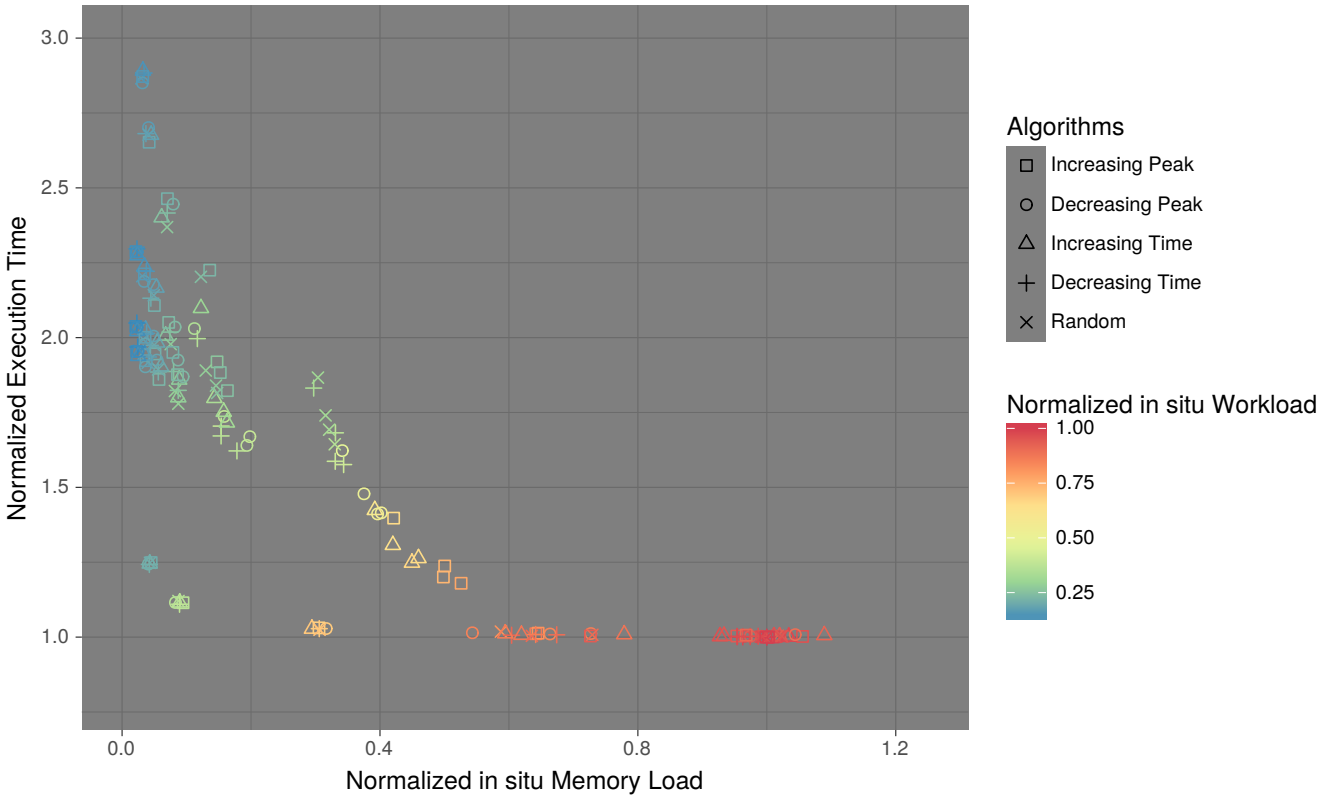

Figure 7: Evaluation of *in situ* memory and workload parameters in algorithm performance for asynchronous scenario.

representing its normalized *in situ* workload.

One should read the figure as follows. In a first time, we are interested at looking where most of the points are located regarding x-axis. Indeed, if a non negligible number points are greater than 1 on this axis, it means that the optimal solution does not have a strong correlation with *in situ* memory usage. We clearly see that at the exception of some points, most of the points have a value less than 1. The exception points stand for cases with lowest application memory load. Hence, the optimal solution is strongly correlated to *in situ* memory usage. This confirms the previous intuition. Let us now take a look on the y-axis and the coloring corresponding to *in situ* workload. We clearly see a trend between the performance of algorithm and the *in situ* workload. Indeed, the best performance are obtained when the *in situ* workload is closed to optimal, hence indicating a correlation in optimal solution between performance and *in situ* workload. All features together, this plot show the strong correlation between *in situ* resource usage and algorithm performance.

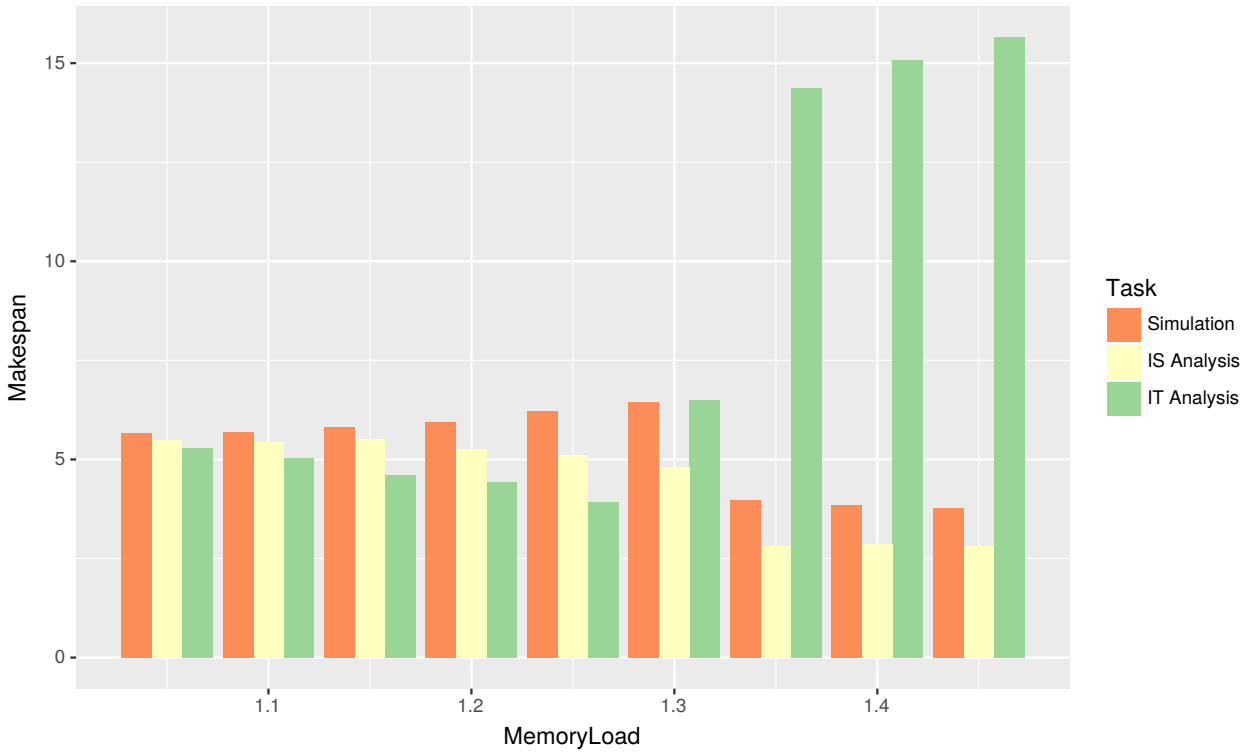

Figure 8: Comparison of simulation, *in situ* and *in transit* analysis makespans for different application memory load for Optimal solution when bandwidth per node equals to 10% of memory per node.

Finally, Figure 8 shows the evolution of the three different makespans (simulation and analysis) of optimal solution for a bandwidth at 10% of memory per node. An interesting remark is that the optimal solution is often a mixture of *in situ*/*in transit*. We see that the simulation is always the larger makespan until the memory load of application overcomes system capacity. This comes from the rounding strategy discussed in Section 4.3. Once memory load of application reaches the system memory capacity, the analysis have to be offloaded *in transit*, thus generating a communication overhead as we can see in the figure.

Figures 8-9 show different trends for random and Increasing Peak heuristic[6]. Due to their greedy behavior, those heuristics generate a schedule with heavy load of *in transit* analytics, hence important transfer overhead. Thus, the *in transit* analytics makespan is the main cost of each iteration for greedy heuristics, explaining the gap of performance with optimal solution.

[6]The trends are similar for other heuristics and different bandwidths.

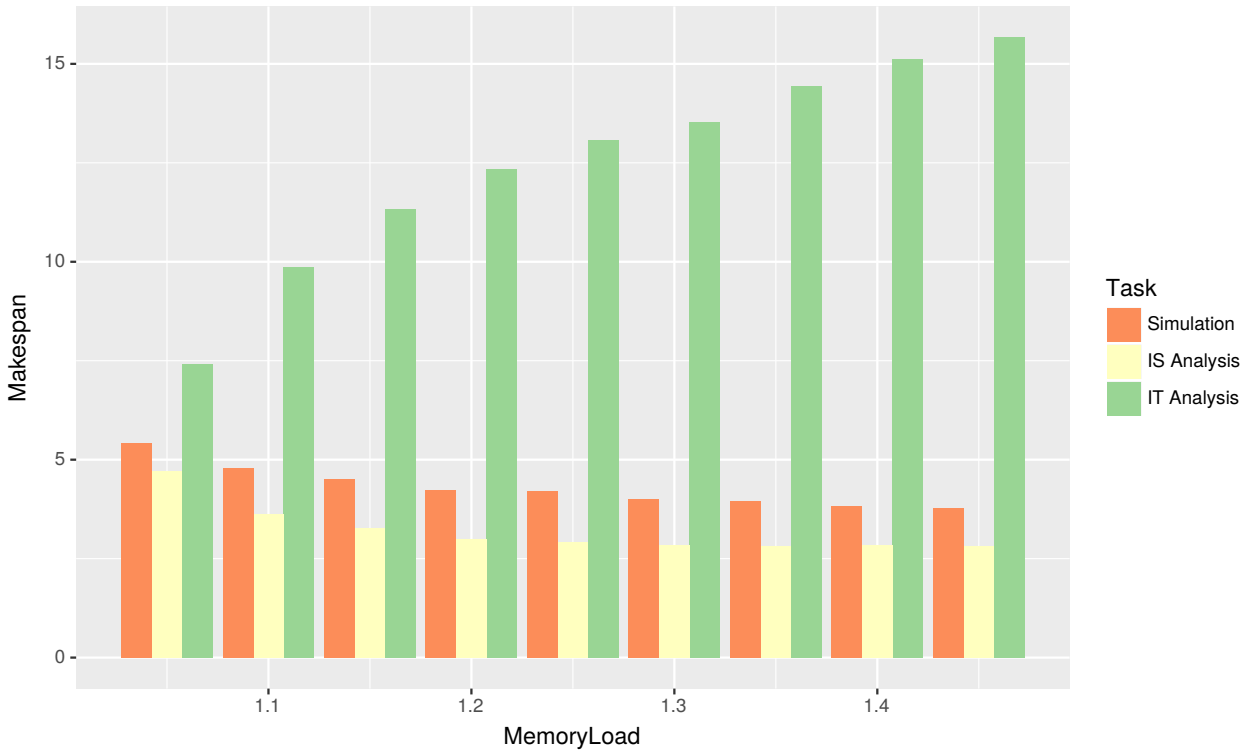

Figure 9: Comparison of simulation, *in situ* and *in transit* analysis makespans for different application memory load for Increasing Peak heuristic when bandwidth per node equals to 10% of memory per node in asynchronous scenario.

6.4 Results for Synchronous scenario

In this section, we compare the performance of asynchronous versus synchronous analytics. We expect the synchronous scenario to outperform the asynchronous one. Recall that in synchronous, we iterate the simulation over all *in situ* cores, then we pause it to perform the *in situ* analysis on the same cores as the *in transit* analytics on its dedicated nodes. In the latter scenario, the working surface dedicated to simulation and analysis is more important than the one in the asynchronous case, thus induces better performance. In practice, synchronous analytics causes complex side effects such as cache management cost during the switch between tasks, so is intrusive to the code. Moreover, simulation is often not able to efficiently scale while we increase the number of cores. A task model such as Amdahl's law [Amd67] would even reinforce this effect. From this perspective, asynchronous analytics is more beneficial despite the data copies it engenders. This would be the target of a future work. For now, the synchronous scenario represents a lower bound on application performance as we

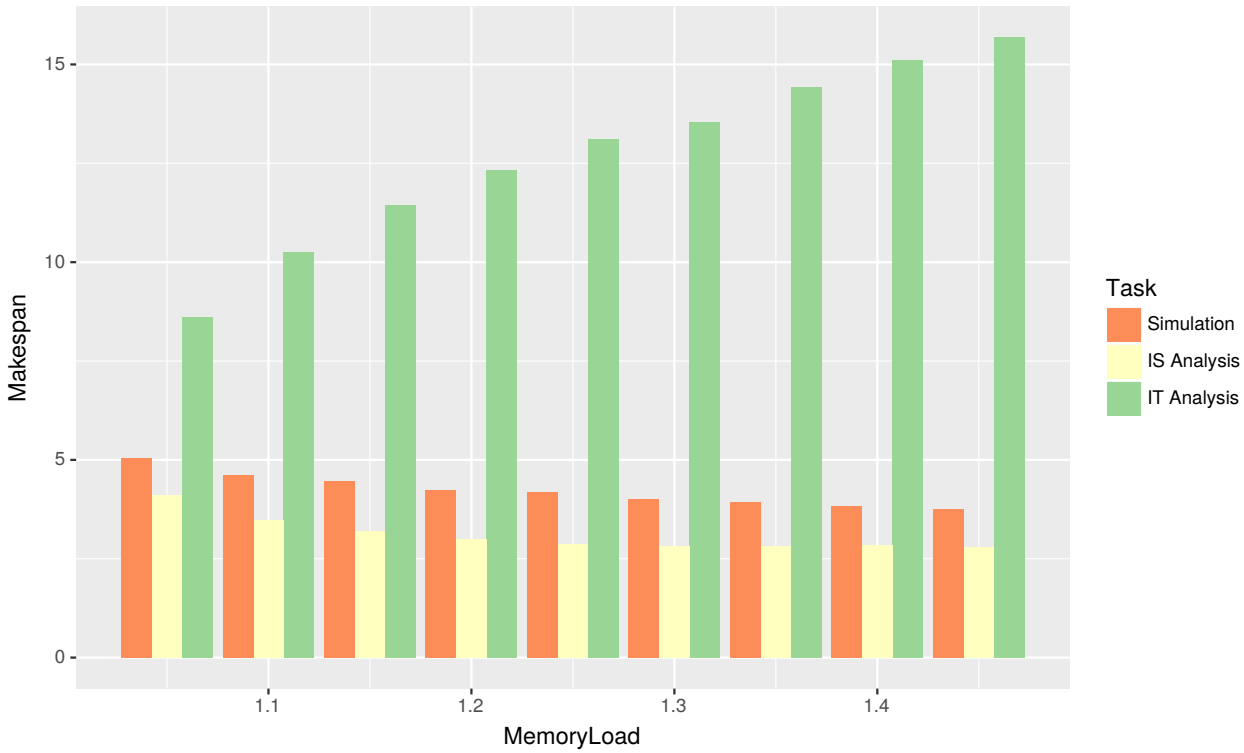

Figure 10: Comparison of simulation, *in situ* and *in transit* analysis makespans for different application memory load for Random heuristic when bandwidth per node equals to 10% of memory per node.

consider embarrassingly parallel tasks.

Figures 11-12 show the simulation results in synchronous case for a setup similar to the one in Section 6.2. We plot in those figures the two best algorithms (Decreasing Peak and Increasing Time[7]) and also the optimal solution of asynchronous scenario, all normalized by the optimal synchronous solution. We see that two optimal solutions have at most a 10% difference and tend to be very similar when the application memory load overflows system capacity. We observe for the greedy heuristics that the synchronous mode does not help to enhance performance. Some difference may occur in some cases, but never more than a 10% gap between the performance of a heuristic in synchronous/asynchronous processing. With a bigger number of cores per node, we know that this difference will tend to reduce, due to the flexibility the increasing number of cores provides.

[7]The order of performance for all other heuristics in synchronous mode is the same as in the asynchronous results.

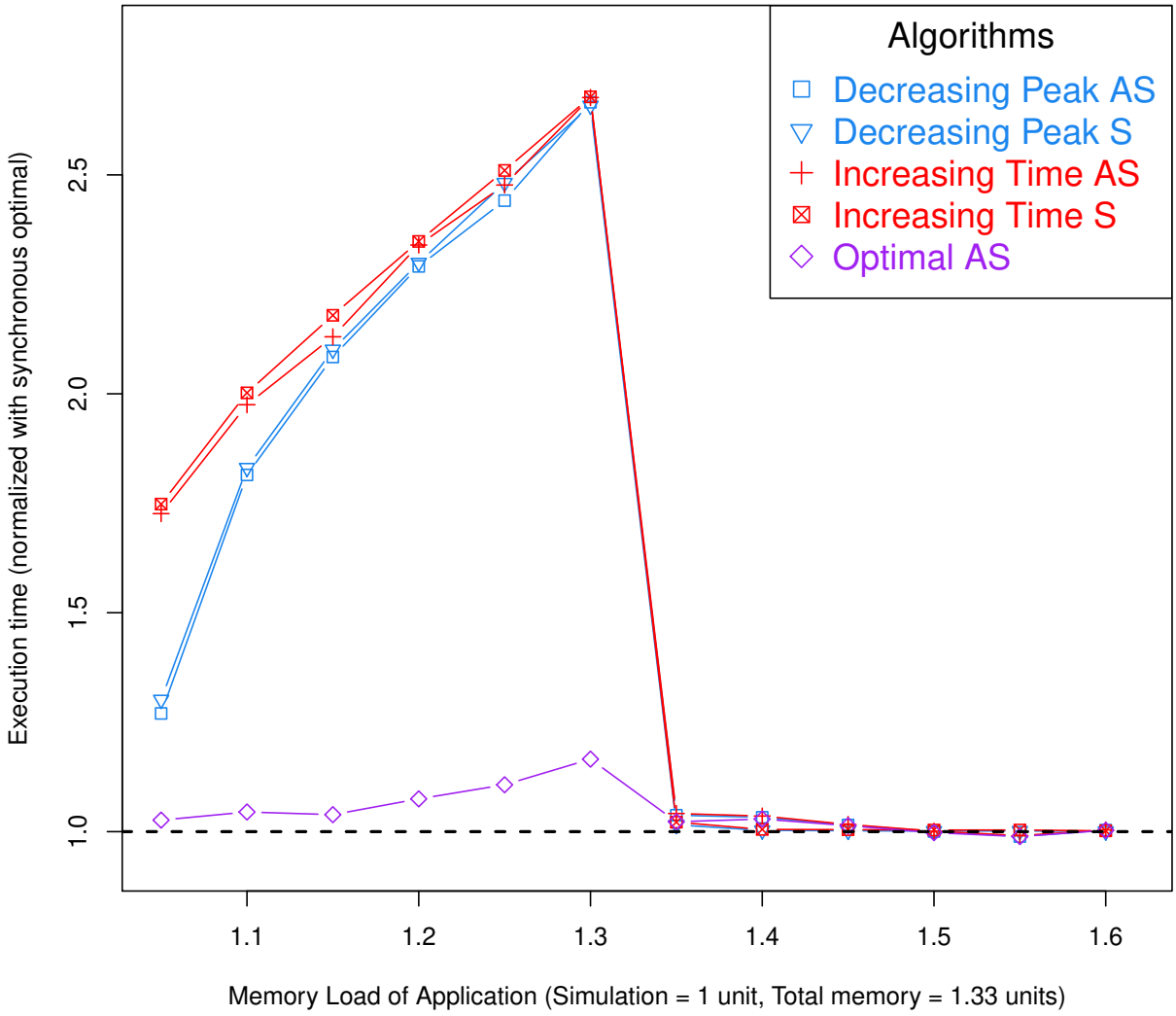

Figure 11: Simulation of Algorithm performance in synchronous scheme with bandwidth is equal to 25% of the memory per node, per unit time.

We see the benefits of synchronous execution mode for *in situ* processing when we have *in situ* analytics to process. However, when the application memory load

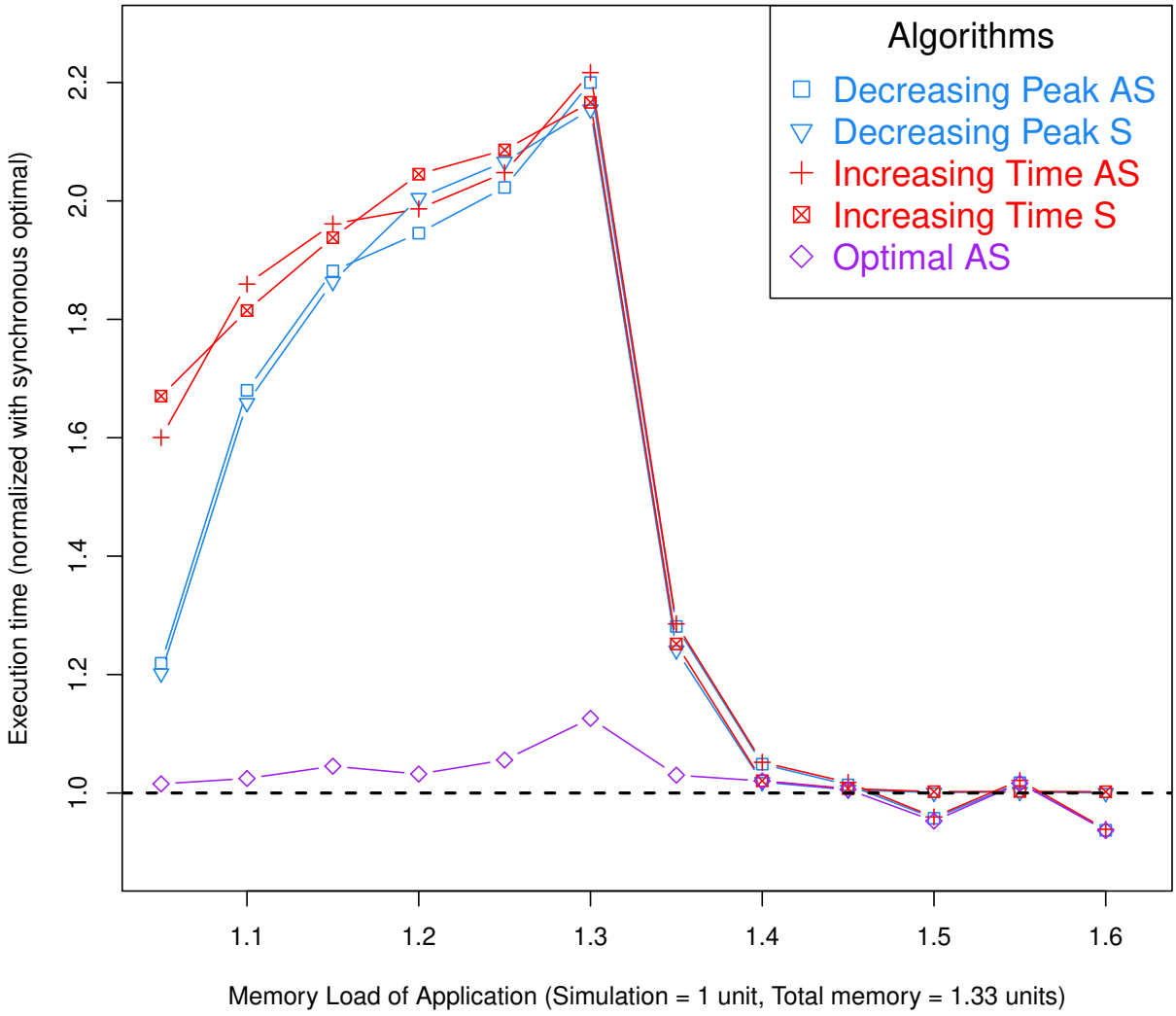

Figure 12: Simulation of Algorithm performance in synchronous scheme with bandwidth is equal to 100% of the memory per node, per unit time.

rises, we see that the optimal synchronous can be outperformed by the optimal asynchronous. This is due to the fact that, in synchronous, there is not enough *in situ*

work to take the benefits of all *in situ* resources when the simulation is paused. In contrast, asynchronous scenario induces a more flexible resource partitioning with less *in situ* resource loss.

7 Conclusion and Future Work

In this paper, we consider the problem of scheduling analysis functions with high-performance applications. This problem is two-fold: it consists in partitioning correctly the resources shared between the simulation and the analysis (nodes, memory etc), and scheduling the different analysis in order to perform them in situ or in transit.

In order to do this, we proposed a first model for the analysis pipeline. This model relies on several assumptions that we have either tried to keep as inconsequential as possible, or that are easily modifiable (e.g. bandwidth model, task model) in our analysis. Based on this model, we have designed several scheduling policies which we have evaluated through intensive simulation. We were able to assert that the memory usage of analysis functions seems to be one of the key feature to account for when performing the scheduling of analysis functions. Specifically, when partitioning analysis functions between *in situ* and *in transit*, one needs to maximize the amount of analysis functions computed *in situ*.

Future work will be dedicated to evaluating experimentally this result. We want to study how robust the assumptions made by our model are, and if our algorithms can be used as such, or if we need to make our model more precise.

Once this is done, we can focus on designing algorithms that maximize the usage of *in situ* resources (memory and cores). A first idea would be to consider the analysis by packs rather than one by one. In general, we would like to quantify the bound that makes *in situ* analysis interesting from a performance perspective.

Several other directions include enriching our model by considering heterogeneous nodes, hierarchical memory and different communication models. In general, we would like to remove some strong assumptions such that perfectly parallel tasks and investigate about parallelization of tasks where the speedup is not linear.

Finally, we will also need to consider a more intricate analysis model, where there are dependencies between analysis functions, and where the results of some analysis modify the simulation steps. New scheduling policies will need to be introduced to consider this directed task-graph model.

Acknowledgements

The authors would like to thank the reviewers for their thorough work.

References

[ABH+13] Guillaume Aupy, Anne Benoit, Thomas Hérault, Yves Robert, Frédéric Vivien, and Dounia Zaidouni. On the combination of silent error detection and checkpointing. In *IEEE 19th Pacific Rim International Symposium on Dependable Computing, PRDC 2013, Vancouver, BC, Canada, December 2-4, 2013*, pages 11–20, 2013.

[AEW+11] Hasan Abbasi, Greg Eisenhauer, Matthew Wolf, Karsten Schwan, and Scott Klasky. Just in Time: Adding Value to the IO Pipelines of High Performance Applications with JITStaging. In *International symposium on High performance distributed computing*, pages 27–36, 2011.

[Amd67] G. Amdahl. The validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, pages 483–485, 1967.

[ARVZ14] Guillaume Aupy, Yves Robert, Frédéric Vivien, and Dounia Zaidouni. Checkpointing algorithms and fault prediction. *J. Parallel Distrib. Comput.*, 74(2):2048–2064, 2014.

[ATNW11] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 23(2):187–198, 2011.

[AWE+09] Hasan Abbasi, Matthew Wolf, Greg Eisenhauer, Scott Klasky, Karsten Schwan, and Fang Zheng. DataStager: Scalable Data Staging Services for Petascale Applications. In *18th ACM international symposium on High performance distributed computing*, pages 39–48, 2009.

[AWW+16] Utkarsh Ayachit, Brad Whitlock, Matthew Wolf, Burlen Loring, Berk Geveci, David Lonie, and E. Wes Bethel. The sensei generic in situ interface. In *Proceedings of the 2Nd Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization*, ISAV '16, pages 40–44, Piscataway, NJ, USA, 2016. IEEE Press.

[BAB+12] 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.

[BSO+11] John Biddiscombe, Jerome Soumagne, Guillaume Oger, David Guibert, and Jean-Guillaume Piccinali. Parallel Computational Steering and Analysis for HPC Applications using a ParaView Interface and the HDF5 DSM Virtual File Driver. In Torsten Kuhlen, Renato Pajarola, and Kun Zhou, editors, *Eurographics Symposium on Parallel Graphics and Visualization*, pages 91–100, 2011. Honourable Mention Award.

[BTSA12] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. Legion: Expressing locality and independence with logical regions. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 66:1–66:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[CML18] Julien Capul, Sébastien Morais, and Jacques-Bernard Lekien. Padawan: A python infrastructure for loosely coupled in situ workflows. In *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ISAV '18, pages 7–12, New York, NY, USA, 2018. ACM.

[DAC+12] Matthieu Dorier, Gabriel Antoniu, Franck Cappello, Marc Snir, and Leigh Orf. Damaris: How to Efficiently Leverage Multicore Parallelism to Achieve Scalable, Jitter-free I/O. In *CLUSTER - IEEE International Conference on Cluster Computing*, Beijing, China, September 2012. IEEE.

[DCR18] Estelle Dirand, Laurent Colombet, and Bruno Raffin. TINS: A Task-Based Dynamic Helper Core Strategy for In Situ Analytics. In *SCA18 - Supercomputing Frontiers Asia 2018*, Singapore, Singapore, March 2018.

[DP16] Matthieu Dreher and Tom Peterka. Bredala: Semantic Data Redistribution for In Situ Applications. In *Proceedings of IEEE Cluster 2016*. IEEE, 2016.

[DPK08] Ciprian Docan, Manish Parashar, and Scott Klasky. Dart: a substrate for high speed asynchronous data io. In *17th international symposium on High performance distributed computing*, pages 219–220, 2008.

[DPK12] Ciprian Docan, Manish Parashar, and Scott Klasky. DataSpaces: an Interaction and Coordination Framework for Coupled Simulation Workflows. *Cluster Computing*, 15(2):163–181, 2012.

[DR14] M. Dreher and B. Raffin. A flexible framework for asynchronous in situ and in transit analytics for scientific simulations. In *2014 14th*

IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pages 277–286, May 2014.

[DSP+13] Matthieu Dorier, Roberto Sisneros, Tom Peterka, Gabriel Antoniu, and Dave Semeraro. Damaris/Viz: a Nonintrusive, Adaptable and User-Friendly In Situ Visualization Framework. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, 2013.

[DSS+05] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Sci. Program.*, 13(3):219–237, July 2005.

[FMT+11] N. Fabian, K. Moreland, D. Thompson, A. C. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. E. Jansen. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *2011 IEEE Symposium on Large Data Analysis and Visualization*, pages 89–96, Oct 2011.

[HHBD17] Reazul Hoque, Thomas Herault, George Bosilca, and Jack Dongarra. Dynamic Task Discovery in PaRSEC: A Data-flow Task-based Runtime. In *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ScalA '17, pages 6:1–6:8, New York, NY, USA, 2017. ACM.

[HMM14] Tim Harris, Martin Maas, and Virendra J. Marathe. Callisto: Co-scheduling parallel runtime systems. In *Proceedings of the Ninth European Conference on Computer Systems (EuroSys'14*, pages 24:1–24:14, 2014.

[HS11] Maurice Herlihy and Nir Shavit. *The art of multiprocessor programming.* Morgan Kaufmann, 2011.

[HSP+17] A. Heirich, E. Slaughter, M. Papadakis, W. Lee, T. Biedert, and A. Aiken. In situ visualization with task-based parallelism. In *Workshop on In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization (ISAV'17)*, 2017.

[KHAL+14] Hartmut Kaiser, Thomas Heller, Bryce Adelstein-Lelbach, Adrian Serio, and Dietmar Fey. Hpx: A task based programming model in a global address space. In *Proceedings of the 8th International Conference*

on Partitioned Global Address Space Programming Models (PGAS'14), 2014.

[LAA+17] Matthew Larsen, James Ahrens, Utkarsh Ayachit, Eric Brugger, Hank Childs, Berk Geveci, and Cyrus Harrison. The alpine in situ infrastructure: Ascending from the ashes of strawman. In *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization*, pages 42–46. ACM, 2017.

[LAB+06] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific Workflow Management and the Kepler System: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, August 2006.

[LHK+16] Matthew Larsen, Cyrus Harrison, James Kress, David Pugmire, Jeremy S. Meredith, and Hank Childs. Performance Modeling of In Situ Rendering. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC16)*, pages 24:1–24:12, Salt Lake City, UT, November 2016.

[LKS+08] Jay F. Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. Flexible io and integration for scientific codes through the adaptable io system (adios). In *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, CLADE '08, pages 15–24, New York, NY, USA, 2008. ACM.

[LVB+10] Min Li, Sudharshan S. Vazhkudai, Ali R. Butt, Fei Meng, Xiaosong Ma, Youngjae Kim, Christian Engelmann, and Galen Shipman. Functional partitioning to optimize end-to-end performance on many-core architectures. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2010.

[MDRP17] Clement Mommessin, Matthieu Dreher, Bruno Raffin, and Tom Peterka. Automatic data filtering for in situ workflows. In *IEEE Cluster*, Hawaii, 2017.

[MLW06] Xiaosong Ma, J. Lee, and M. Winslett. High-Level Buffering for Hiding Periodic Output Cost in Scientific Simulations. *Parallel and Distributed Systems, IEEE Transactions on*, 17(3):193–204, 2006.

[MVK+16] Preeti Malakar, Venkatram Vishwanath, Christopher Knight, Todd Munson, and Michael E. Papka. Optimal execution of co-analysis for

large-scale molecular dynamics simulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, pages 60:1–60:14, Piscataway, NJ, USA, 2016. IEEE Press.

[MVM+15] Preeti Malakar, Venkatram Vishwanath, Todd Munson, Christopher Knight, Mark Hereld, Sven Leyffer, and Michael E. Papka. Optimal scheduling of in-situ analysis for large-scale scientific simulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 52:1–52:11, New York, NY, USA, 2015. ACM.

[MWYT07] Kwan-Liu Ma, Chaoli Wang, Hongfeng Yu, and Anna Tikhonova. In-situ processing and visualization for ultrascale simulations. *Journal of Physics: Conference Series*, 78(1):012043, 2007.

[PBH+16] P. Pébaÿ, J. C. Bennett, D. Hollman, S. Treichler, P. S. McCormick, C. M. Sweeney, H. Kolla, and A. Aiken. Towards asynchronous many-task in situ data analysis using legion. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1033–1037, May 2016.

[SBF09] Ajeet Singh, Pavan Balaji, and Wu-chun Feng. GePSeA: A General-Purpose Software Acceleration Framework for Lightweight Task Offloading. In *International Conference on Parallel Processing*, pages 261–268, 2009.

[SDD+18] Pradeep Subedi, Philip Davis, Shaohua Duan, Scott Klasky, Hemanth Kolla, and Manish Parashar. Stacker: an autonomic data movement engine for extreme-scale data staging-based in situ workflows. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'18)*, page 73. IEEE Press, 2018.

[SJR+15] Qian Sun, Tong Jin, Melissa Romanus, Hoang Bui, Fan Zhang, Hongfeng Yu, Hemanth Kolla, Scott Klasky, Jacqueline Chen, and Manish Parashar. Adaptive data placement for staging-based coupled scientific workflows. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 65:1–65:12, New York, NY, USA, 2015. ACM.

[TDCC17] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *2017 IEEE*

International Parallel and Distributed Processing Symposium, IPDPS 2017, Orlando, FL, USA, May 29 - June 2, 2017, pages 1129–1139, 2017.

[TRB+08] Tiankai Tu, Charles A. Rendleman, David W. Borhani, Ron O. Dror, Justin Gullingsrud, Morten Ø. Jensen, John L. Klepeis, Paul Maragakis, Patrick Miller, Kate A. Stafford, and David E. Shaw. A Scalable Parallel Framework for Analyzing Terascale Molecular Dynamics Simulation Trajectories. In *Conference on Supercomputing*, pages 56:1–56:12, 2008.

[VHP11] V. Vishwanath, M. Hereld, and M.E. Papka. Toward simulation-time data analysis and i/o acceleration on leadership-class systems. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 9–14, Oct 2011.

[WABJ15] Yi Wang, Gagan Agrawal, Tekin Bicer, and Wei Jiang. Smart: A mapreduce-like framework for in-situ scientific analytics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 51:1–51:12, New York, NY, USA, 2015. ACM.

[WFM11] Brad Whitlock, Jean M. Favre, and Jeremy S. Meredith. Parallel in situ coupling of simulation with a fully featured visualization system. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization*, EGPGV '11, pages 101–109, Aire-la-Ville, Switzerland, Switzerland, 2011. Eurographics Association.

[WWP09] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[YWG+10] Hongfeng Yu, Chaoli Wang, R.W. Grout, J.H. Chen, and Kwan-Liu Ma. In situ visualization for large-scale combustion simulations. *Computer Graphics and Applications, IEEE*, 30(3):45–57, 2010.

[ZAD+10] Fang Zheng, H. Abbasi, C. Docan, J. Lofstead, Qing Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. PreDatA - Preparatory Data Analytics on Peta-Scale Machines. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12, 2010.

[ZDP+12] Fan Zhang, C. Docan, M. Parashar, S. Klasky, Norbert Podhorszki, and Hasan Abbasi. Enabling In-situ Execution of Coupled Scientific Workflow on Multi-core Platform. In *Parallel Distributed Processing Symposium (IPDPS)*, pages 1352–1363, 2012.

[ZYH+13] F. Zheng, H. Yu, C. Hantas, M. Wolf, G. Eisenhauer, K. Schwan, H. Abbasi, and S. Klasky. Goldrush: Resource efficient in situ scientific data analytics using fine-grained interference aware execution. In *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12, Nov 2013.

[ZZE+13] F. Zheng, H. Zou, G. Eisenhauer, K. Schwan, M. Wolf, J. Dayal, T. Nguyen, J. Cao, H. Abbasi, S. Klasky, N. Podhorszki, and H. Yu. Flexio: I/o middleware for location-flexible scientific data analytics. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 320–331, May 2013.