DataStorm: Coupled, Continuous Simulations for Complex Urban Environments

HANS WALTER BEHRENS, K. SELÇUK CANDAN, XILUN CHEN, YASH GARG, MAO-LIN LI, XINSHENG LI, and SICONG LIU, Arizona State University
MARIA LUISA SAPINO, University of Turin
MD SHADAB, DALTON TURNER, and MAGESH VIJAYAKUMAREN, Arizona State University

Urban systems are characterized by complexity and dynamicity. Data-driven simulations represent a promising approach in understanding and predicting complex dynamic processes in the presence of shifting demands of urban systems. Yet, today's silo-based, de-coupled simulation engines fail to provide an end-to-end view of the complex urban system, preventing informed decision-making. In this article, we present `DataStorm` to support integration of existing simulation, analysis and visualization components into integrated workflows. `DataStorm` provides a flow engine, `DataStorm-FE`, for coordinating data and decision flows among multiple actors (each representing a model, analytic operation, or a decision criterion) and enables ensemble planning and optimization across cloud resources. `DataStorm` provides native support for simulation ensemble creation through parameter space sampling to decide which simulations to run, as well as distributed instantiation and parallel execution of simulation instances on cluster resources. Recognizing that simulation ensembles are inherently sparse relative to the potential parameter space, we also present a density-boosting partition-stitch sampling scheme to increase the effective density of the simulation ensemble through a subspace partitioning scheme, complemented with an efficient stitching mechanism that leverages partial and imperfect knowledge from partial dynamical systems to effectively obtain a global view of the complex urban process being simulated.

CCS Concepts: • **Computing methodologies** → **Continuous simulation**; **Simulation support systems**; • **Information systems** → *Spatial-temporal systems*; *Data provenance*;

Additional Key Words and Phrases: Simulation ensembles, urban computing, simulation space sampling

ACM Reference format:
Hans Walter Behrens, K. Selçuk Candan, Xilun Chen, Yash Garg, Mao-Lin Li, Xinsheng Li, Sicong Liu, Maria Luisa Sapino, Md Shadab, Dalton Turner, and Magesh Vijayakumaren. 2021. DataStorm: Coupled, Continuous Simulations for Complex Urban Environments. *ACM/IMS Trans. Data Sci.* 2, 3, Article 19 (April 2021), 37 pages.
https://doi.org/10.1145/3447572

Authors are listed alphabetically. This is an extended version of the conference paper [29].

This research is supported by NSF#1610282 "DataStorm: A Data Enabled System for End-to-End Disaster Planning and Response," NSF#1633381 "BIGDATA: Discovering Context-Sensitive Impact in Complex Systems," NSF#1909555 "pCAR: Discovering and Leveraging Plausibly Causal (p-causal) Relationships to Understand Complex Dynamic Systems," and "FourCmodeling": EUH2020 Marie Sklodowska-Curie grant agreement No 690817. Results were obtained using the Chameleon testbed [22] supported by the NSF.

Authors' addresses: H. W. Behrens, K. S. Candan, X. Chen, Y. Garg, M.-L. Li, X. Li, S. Liu, MD Shadab, D. Turner, and M. Vijayakumaren, Arizona State University, 699 S Mill Ave, Tempe, Arizona 85281-3673, USA; emails: {hwb, candan, xchen120, ygarg, maolinli, lxinshen, sliu104, mshadab1, dcturne4, mvijaya4}@asu.edu.

2577-3224/2021/04-ART19 $15.00
https://doi.org/10.1145/3447572

1 INTRODUCTION

Urban systems are characterized by high complexity and dynamicity, requiring data- and model-driven planning. In 2016, for example, flooding in Texas caused residents to evacuate their homes with only hours of warning, despite hurricane simulations accurately predicting the path of the storm weeks in advance. Unfortunately, rainfall predictions from the hurricane simulations could not be fed directly into hydrographic models that predicted water flows, leading to a failure in preparation. This is a major challenge due to overlapping and cascading processes, especially when involving multi-hazard scenarios where one hazard (e.g., flooding) is the gateway to the next (e.g., an epidemic). Yet, today's *silo-based, de-coupled simulation engines* assume that disaster, population dynamics, transportation, and disease/epidemic simulations are not integrated, failing to provide an end-to-end view for informed decision-making.

1.1 Data- and Model-driven Urban Modeling and Disaster Management

Data-driven models and computer simulations that take into account available domain knowledge, in addition to past data and observations, represent a promising approach in understanding and predicting complex urban processes [4, 15, 33, 45] and in information-driven decision-making in the presence of shifting demands and available resources [8]. Therefore, data-driven models and simulations for preparedness and response are increasingly critical in predicting geo-temporal evolution of urban systems and effectively managing emergencies through intervention measures [10, 11, 31, 32, 35, 39]. However, existing methods have two fundamental limitations:

(a) First, to be useful, simulations need to track hundreds or thousands of inter-dependent parameters, spanning multiple layers and geo-spatial frames and affected by complex natural and urban processes operating at different resolutions (Figure 1(a)). Moreover, many of these dynamically evolve over time, requiring continuous adaptation and re-modeling (Figure 1(b)).

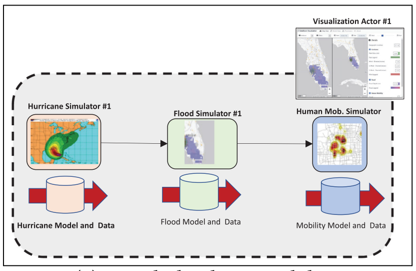
(b) Second, urban processes involve various distinct yet inter-dependent processes (e.g., population movement/evacuations, epidemics that result from poor living conditions, and/or population concentration and mobility) that operate at varying spatial and temporal scales; currently, simulations in individual domains are performed in an isolated manner. In contrast, real-time and continuous analysis and decision-making, including forecasting the response of the affected urban and/or rural populations and predicting their pre-, during-, and post-disaster movement, and assessing the effect of evacuations and/or travel controls and other interventions before, during, and after a disaster require integration and analysis of large volumes of (weather, demographic, road networks, and disease) data and models.

1.2 Data Challenges: Inherent Sparsity of Simulation Ensembles

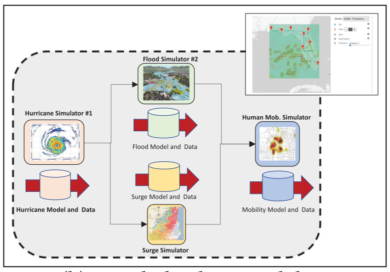
In this article, we introduce a novel data and computing infrastructure, named `DataStorm` [5], that supports seamless integration of independently developed, reusable scientific models and analysis components. `DataStorm` solves a wide variety of challenges, including data interoperability, scalability, temporal and spatial alignment, and sparse simulation space sampling, among others:

Complexity and Heterogeneity: Simulations need to track hundreds of interdependent parameters, spanning multiple geospatial frames, affected by complex inter-dependent dynamic processes operating at different resolutions. Yet, today's silo-based, de-coupled simulation engines assume that disaster, population dynamics, transportation, and disease/epidemic simulations are not integrated, failing to provide an end-to-end view of the disaster.

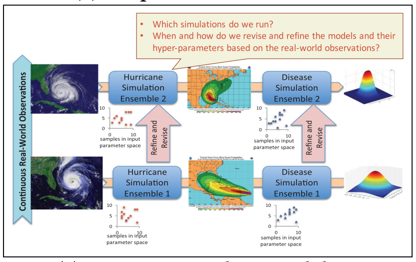
Dynamicity: In addition, as the data arrives in a streaming fashion, simulation ensembles need to be continuously revised and refined as the situation on the ground changes: (a) revisions involve incorporating real-world observations as well as updated probability densities into existing

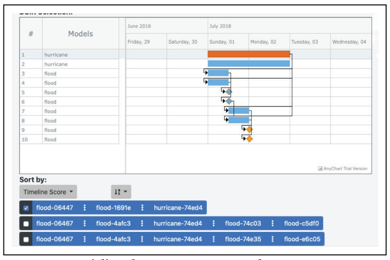


(a) coupled urban model #1



(b) coupled urban model #2


Which simulations do we run?
When and how do we revise and refine the models and their hyper-parameters based on the real-world observations?
Continuous Real-World Observations
Hurricane Simulation Ensemble 2
Disease Simulation Ensemble 2
Hurricane Simulation Ensemble 1
Disease Simulation Ensemble 1
Refine and Revise
Refine and Revise
samples in input parameter space
samples in input parameter space
samples in input parameter space
samples in input parameter space
(c) continuous urban modeling



(d) alternate timelines

Fig. 1. Coupled, continuous urban modeling: (a,b) DataStorm enables plug-and-play execution of coupled simulations, where the user can change any of the component simulators, add new simulation components, or alter the visualization strategy; (c) moreover, the models are executed continuously, leading to (d) alternate timelines (i.e., possible outcomes under different assumptions) that can be explored by the users.

simulations to alter their outcomes; (b) refinements include identifying new simulations to run, incorporating the changing situation on the ground to support improved decision-making.

Sparsity: Simulation ensembles are *inherently sparse*. Due to the large number of unknowns, decision-makers usually need to generate ensembles of stochastic scenarios, requiring hundreds or thousands of individual simulation instances, each with different parameter settings corresponding to distinct plausible scenarios. Moreover, as the number of input parameters of a simulation increases, the number of potential situations one can simulate increases exponentially. Therefore, to support effective decision-making, one must answer the question, "Given a large parameter space and fixed resources and time, which simulation instances should be included?" Yet, there is little support for simulation ensemble planning and optimization.

1.3 DataStorm: Contributions of the Article

To address the above challenges, we introduce DataStorm, a cloud-based data infrastructure capable of executing multiple coupled simulations synergistically, under a unified model; addressing computational challenges that arise from the need to acquire, integrate, model, analyze, index, and search, in a scalable manner, large volumes of multi-variate, multi-layer, multi-resolution, interconnected, and inter-dependent spatio-temporal data that arise from urban simulations and real-world observations; and a high-performance data-processing system to support coupled, continuous observation of the numerical results for simulations from different domains with diverse resource demands and constraints. The novel DataStorm system aims to improve the impact of community-supplied data and models through sharing and integration of existing, reusable analysis, data processing, and visualization components into integrated simulation/analysis workflows and provides a spectrum of services, including (a) model/data integration and alignment, (b) distributed/parallel workflow orchestration, (c) parameter space sampling and simulation ensemble creation, and (d) ensemble storage, search, analysis, visualization, and sharing.

In this article, to address the sparsity challenge, we propose a *density boosting partition-stitch sampling* scheme, which divides the ensemble construction process into the task of constructing lower modal sub-ensembles. Based on the observation that while most complex processes can be partitioned to capture different sub-processes, these partitions nevertheless relate to each other and, hence, reflect the footprints of the same underlying global pattern, we propose to partition the given system into multiple sub-systems, analyzed independently. To *transfer, share, and combine* what we independently learned from the analysis of the sub-systems, we further propose a novel ***multi-task tensor decomposition*** **(M2TD)** technique, which effectively and efficiently stitches these sub-ensembles back, in a way that increases the *effective density* of the simulation samples to boost accuracy. Experiment results presented in Section 5 show that, for a given budget of simulations, the proposed density boosting partition-stitch sampling scheme leads to significantly better accuracy relative to traditional sampling approaches.

2 RELATED WORK

As discussed in the introduction, simulations are being increasingly used for understanding and predicting complex urban processes [4, 15, 33, 45], including in applications including disaster preparedness and response [8, 10, 11, 31, 32, 35, 39]. In this section, we provide an overview of some of the key technologies underlying complex coupled simulation and decision flow systems.

2.1 Scientific Workflow Systems

Scientific workflow systems, such as Kepler [1] and Taverna [18], provide tools for structured, repeatable data generation and transformation tasks. However, existing systems provide limited support for uncertainty, imprecision, or adaptation, such as that required by urban data/decision workflows. In general, such systems do not consider scenarios requiring continuous execution, preventing workflows operating on data streams. CONFLuEnCE [34] builds upon Kepler to add continuous execution support, but does not address changes to the underlying workflow in response to such streams. Existing systems often assume serializable or single-system workflow execution, posing significant scalability challenges. BioKepler [2] addresses this through distributed, cluster-based execution, but a strong focus on biological simulations limits broader applicability. To the best of our knowledge, no existing scientific workflow systems provide adaptive workflow rewriting, where workflow structure, behavior, or data flow changes in response to current results.

2.2 Scientific Computation Platforms

Scientific computation, especially that related to complex domain-specific simulations, often requires significant resource allocations, specialized execution characteristics, or other deployment constraints. To address these considerations, several large-scale platforms have been created for use by the open research community, such as the NSF-supported Chameleon Cloud [22]. These systems leverage cloud technologies to provide scalable, configurable, and performant platforms suitable for scientific computation. Control extends from bare metal configuration up to fully functional cloud environments by leveraging open-source tools such as OpenStack and Atmosphere. However, these platforms are designed to support a broad range of computational experiments involving interactive computing and data analysis. Thus, they do not provide suitable interfaces for direct integration with scientific computation. The *experiment précis* offered by Chameleon Cloud records user actions to provide coarse-grained experiment replication, but does not address data provenance, continuous execution, or other desirable characteristics.

2.3 Simulation Ensembles

Simulation workflows link interrelated models, improving the efficacy beyond that offered by executing the constituent models independently. Such systems have previously been explored in the literature. SimDMS [39], for example, examined a scenario linking epidemic transmission models with a variety of different human mobility simulators to evaluate and directly compare the impacts of population movement on epidemic spread. Similarly, WIFIRE [16] couples several wildfire simulators into a cohesive ensemble, providing better predictive accuracy for wildfire spread through the creation of a heterogeneous-model, shared-domain simulation environment.

However, in both of these works, linkages within the simulation ensemble required handcrafting for compatibility. Other systems [41] execute multiple instances of identical simulators in parallel, each with a different parametric configuration, to better cover the potential simulation space. However, this is designed to work only with identical simulators, restricting ensemble diversity. DataStorm supports workflows including heterogeneous ensembles similar to References [16, 39], homogeneous ensembles similar to Reference [41], or a combination of both.

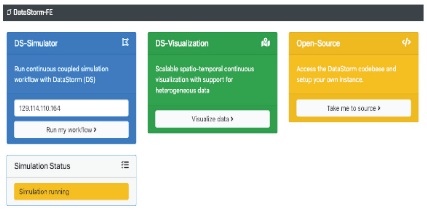
Designing ensemble configurations that appropriately and efficiently cover the input parameter space [7, 37] is not trivial. Work in this area generally falls into two categories: single stage and sequential approaches. Single-stage approaches find their root in design of experiments [14, 21] where, given a simulation budget, samples are allocated in a single iteration to minimize the variance of the subsumed model (such as general linear models and Gaussian processes). Examples are the commonly used Latin hypercube or uniform sampling. These approaches assume a unique response model from throughout the input parameter space. Recent work has also started to take into account budget constraints and costs of configurations during simulation instance selection [37]. Sequential approaches are handled through single- or multiple-run replications [38]. In single-run replication, simulation instances are allocated incrementally, with each step evaluating its performance and deciding the next simulation to run. In multiple-run replication, the parameter space is sampled simultaneously, resulting in multiple, shorter runs. Recent research developments, such as References [7, 37, 38], focus on discrete event simulation output generated either from a long replication of a single run or from multiple shorter replications at the conclusion of execution. In Reference [7], the authors proposed a time dilation scheme, where the units measuring time can be changed at any event. The proposed dilation method penalizes uninteresting or unrealistic parameter settings and allocates more resources to the interesting ones. This is equivalent to biased sampling, where high weights are associated with samples in interesting regions, and correspondingly low weights are assigned to samples from uninteresting parts of the input space. This method suffers from two drawbacks: (a) predicting computational resource consumption can be difficult and (b) the resulting ensemble convergence rate may be slow. In Reference [37], the authors proposed an improvement to address this, which combines time dilation with optimal computing budget allocation. Intuitively, this is similar to identifying interesting parameter settings while simultaneously optimizing the number of replications to run.

3 DATASTORM

In this article, we present DataStorm (source code and services publicly available through open-source licenses at References [42, 43]), a framework for creation, storage, analysis, and exploration of coupled, multi-model urban simulation ensembles.

3.1 Overview of DataStorm and DataStorm Flow Engine (DataStorm-FE)

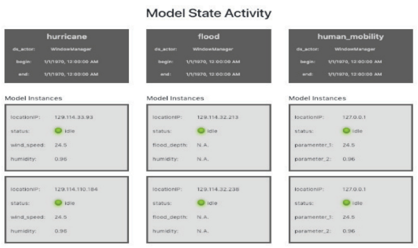
At the core of DataStorm is the DataStorm-FE data- and decision-flow engine and coordination engine for creating and maintaining coupled, multi-model simulation ensembles. DataStorm-FE


DataStorm-FE
DS-Simulator
Run continuous coupled simulation workflow with DataStorm (DS)
Run my workflow ›
DS-Visualization
Scalable spatio-temporal continuous visualization with support for heterogeneous data
Visualize data ›
Open-Source
Access the DataStorm codebase and setup your own instance.
Take me to source ›
Simulation Status

129.114.110.164

Simulation running

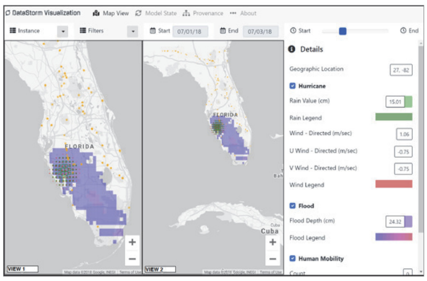
(a) DataStorm cloud interface



hurricane		flood		human_mobility	
ds_actor:	WindowManager	ds_actor:	WindowManager	ds_actor:	WindowManager
begin:	1/1/1970, 12:00:00 AM	begin:	1/1/1970, 12:00:00 AM	begin:	1/1/1970, 12:00:00 AM
end:	1/1/1970, 12:00:00 AM	end:	1/1/1970, 12:00:00 AM	end:	1/1/1970, 12:00:00 AM
locationIP:	129.114.33.93	locationIP:	129.114.32.213	locationIP:	127.0.0.1
status:	● idle	status:	● idle	status:	● idle
wind_speed:	24.5	flood_depth:	N.A.	paramenter_1:	24.5
humidity:	0.96	humidity:	N.A.	parameter_2:	0.96
locationIP:	129.114.110.184	locationIP:	129.114.32.238	locationIP:	127.0.0.1
status:	● idle	status:	● idle	status:	● idle
wind_speed:	24.5	flood_depth:	N.A.	paramenter_1:	24.5
humidity:	0.96	humidity:	N.A.	parameter_2:	0.96

(b) Coupled simulation back-end visualizer

Fig. 2. DataStorm cloud interface and back-end resource visualizer.

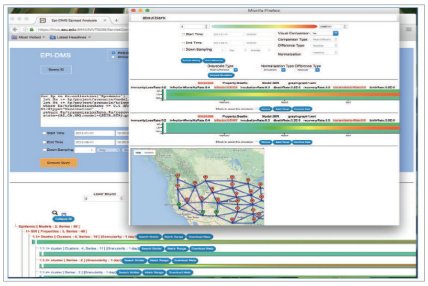


Start 07/01/18 End 07/03/18

Details

Geographic Location	27, -82
✓ Hurricane	
Rain Value (cm)	15.01
Rain Legend	🟩
Wind - Directed (m/sec)	1.06
U Wind - Directed (m/sec)	-0.75
V Wind - Directed (m/sec)	-0.75
Wind Legend	🟥
✓ Flood	
Flood Depth (cm)	24.32
Flood Legend	▨
✓ Human Mobility	
Count	0

(a) DataStorm visualization (integrating hurricane, flood, and mobility models)



(b) Simulation ensemble querying and visualization interface

Fig. 3. DataStorm visualization and ensemble querying.

provides a means for coordinating data- and decision flows among partially connected simulation engines and enables end-to-end ensemble planning and optimization (including parameter-space sampling, output aggregation and alignment, continuous data streaming, and state and provenance data management) to improve the predictive accuracy of the overall end-to-end simulation process within a limited simulation budget. Additionally, DataStorm provides a cloud-based decision support infrastructure for results obtained by DataStorm-FE, permitting users to query, explore, or visualize relevant data to facilitate the decision-making process (Figures 2 and 3).

DataStorm addresses the need to consider multiple distinct, yet inter-dependent models (e.g., hurricane, flood, surge, population dynamics, epidemics) to understand disaster-related processes and to support decision-making (Figure 3(a)). DataStorm-FE coordinates data and decision flows among multiple actors (simulation engines) and enables end-to-end ensemble planning and optimization. Unlike prior scientific workflow engines, DataStorm-FE (a) is designed for ensemble executions, creating not one, but multiple scenarios, simultaneously; (b) is designed for continuous workflow executions with streaming inputs and provides "stateful" DataStorm actors (each actor involves a simulation model, analytic operation, or a decision criterion); and (c) provides native support for parameter space sampling and distributed and parallel execution of instances.

By design, DataStorm is inter-operable with existing tools and cyber-infrastructures. DataStorm is built to interface with leading simulation software, such as GLEaM [3] and STEM [12] for simulations of geotemporal evolution and forecasting of epidemics, EpaNet2, Finesse, H2Onet, and WaterCAD for water distribution, and VISSIM, AIMSIM, SUMO, and others for traffic networks. With the **Weather Research and Forecasting (WRF)** hurricane model [11], we are able to predict the track of a hurricane, as well as associated wind speeds and rainfall. Since this model does not attempt to predict flooding related to these events, we couple the output with Itzï [10], a hydrological simulator that models the flow of flood-waters over the landscape. SLOSH [19] is used for surge modeling. Additionally, the hurricane and floods have behavioral impacts on the affected population; therefore, the corresponding models are coupled with the Opportunistic Network Environment simulator [24], which is used to track population movement through transport networks within the affected area. We note that, while `DataStorm` is primarily conceived to support complex modeling of disasters, the software platform is model-agnostic and can be used as the seed of a more general purpose urban service and compute infrastructure.

At its lowest level, `DataStorm-FE` extends the Kepler scientific workflow system [1], designed to integrate disparate models; in particular, as in Kepler, *actors* provide code modules that execute a particular task, with the data flow between these actors being controlled by a single global *director*. While Kepler provides a basic framework to create executable workflows, including an actor-oriented modeling paradigm, it has significant limitations: (a) Kepler is not designed for ensemble executions; it can only take a fully instantiated model and execute a specified workflow; (b) Kepler does not provide stateful actors and is not designed for continuous workflow executions; and (c) while Kepler's Web and Grid service actors allow scientists to utilize computational resources on the net in a distributed scientific workflow, it does not provide native support for parameter space sampling, distributed instantiation, and parallel execution of simulation instances in an ensemble.

The various components of `DataStorm` are being developed and evaluated under diverse software/hardware settings, including NSF's Chameleon Infrastructure [22] and the NSF-supported GEARS platform that provides heterogeneous computing and storage resources [47]. `DataStorm` uses OpenStack for distributed, parallel simulation instance execution, and Ansible [46] for automated provisioning, configuration management, and deployment. `DataStorm` also relies on MySQL and MongoDB for configuration and provenance/state storage, Apache Tomcat and JQuery for visualization, Python for subcomponent implementation and coordination, and AWS and OpenStack for instance orchestration. User and ensemble management stacks are built on SimDMS [39] for storing and analyzing simulation ensembles (Figure 3(b)).

In the rest of this section, we formalize the `DataStorm` continuous, coupled simulation and decision workflows and describe the `DataStorm-FE` and the `DataStorm` visualization components.

3.2 Models and Data Streams

In `DataStorm-FE`, the inputs to a decision-flow are the data sources, with each step in the decision-flow involving an analytic function, model, or decision criterion. Inputs to each step are data and decisions from the previous steps, plus user-provided decision parameters.

3.2.1 DS-Models.

Definition 1 (Model). A model, $\mathbb{M}$, is a tuple $\langle F, \mathcal{P}, \mathcal{D}, \mathcal{O}, \omega_I, \omega_O, \Delta \rangle$, where $F()$ is a (simulation or analysis) function, $\mathcal{P}$ is a set of simulation parameter names (defining a simulation space), $\mathcal{D}$ is a set of input data variables, and $\mathcal{O}$ is a set of output variables, such that

$$\vec{o} \leftarrow F(\vec{p}, \vec{d}),$$

where $\vec{p}$, $\vec{d}$, and $\vec{o}$ are instantiation vectors for $\mathcal{P}$, $\mathcal{D}$, $\mathcal{O}$, respectively. ◇

Many, if not most, models that describe urban environments or other complex systems make use of temporal information. Whether describing a power grid through demand over time or crime rates in response to enforcement initiatives, the temporal evolution of the environment plays a key role in prediction and analysis. Input and output variables to *F()* have temporal scopes:

- The input variables capture input data to the model for a given input window size, w_{in} (i.e., the temporal input scope is $[t, t + w_{in}))$ for some time instant t; note that in discrete simulations, the input data will also have a discrete resolution, given by res_{in}.
- The output variables capture simulation/model output for a given window size, w_{out}. (i.e., the temporal output scope is $[t, t + w_{out}))$ for some time t; in discrete simulations, the output data will also have a discrete resolution, given by res_{out}.

In the definition of the model, $\omega_I = \langle w_{in}, res_{in} \rangle$ and $\omega_O = \langle w_{out}, res_{out} \rangle$ are the descriptors of the input and output scopes, e.g., the temporal lengths and resolutions for inputs and outputs. Δ is a the value of temporal shift. Since models are temporal, they consume and produce data as streams:

Definition 2 (Input Stream). Let $\mathbb{M}$ be a model. We refer to the sequence $I = (\vec{d}_{(0)}, \vec{d}_{(1)}, \vec{d}_{(2)} \ldots)$, where each $\vec{d}_{(j)}$ is defined in the input data space, $\mathcal{D}$, as an input stream. ◇

Definition 3 (Output Stream). Let $\mathbb{M}$ be a model. Let I be an input stream and $\vec{p}$ be a parameter instantiation, respectively. We refer to the sequence $O = (\vec{o}_{(0)}, \vec{o}_{(1)}, \vec{o}_{(2)} \ldots)$, where each $\vec{o}_{(j)}$ is defined in the output data space, $\mathcal{O}$, as the corresponding output stream. ◇

Since the data available to the models evolve over time, `DataStorm` provides *stateless*, *stateful*, and *memoryful* models:

- In a *stateful* model, there is a special state variable, σ, such that $\vec{d}_{(s+1)}[\sigma] = \vec{o}_{(s)}[\sigma]$; intuitively, this variable encodes the relevant state information from one simulation step to the next.
- In a *memoryful* model, there is a special memory variable, μ, such that $\vec{d}_{(s+1)}[\mu] = \vec{o}_{(s)}[\mu] \odot \vec{d}_{(t)}[\mu]$, where $\odot$ denotes the concatenation operation; intuitively, this variable accumulates and carries the collective memory of the model from one simulation step to the next.

In `DataStorm-FE`, models are implemented within software modules called *DS-Actors*.

3.3 DataStorm Workflows

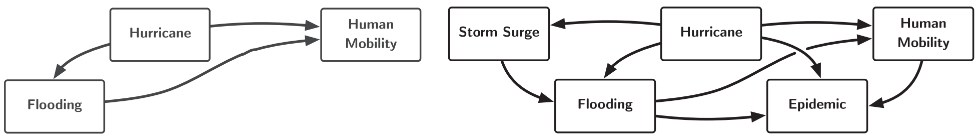
Give the above, a `DataStorm` workflow, or DS-Flow, is defined as follows:

Definition 4 (DS-Flow). A `DataStorm` workflow is a node-labeled directed graph, $G(V, E, l)$, where the label, $l(v_h)$, of node $v_h \in V$ in the graph is a model, $\mathbb{M}_h$—i.e., $l(v_h) = \mathbb{M}_h$. ◇

As stated earlier, the models (and consequently the data- and decision-flows or *DS-Flows*) supported by `DataStorm-FE` are inherently temporal. Therefore, rather than utilizing a single-shot execution, where inputs for the workflow are consumed and actors are invoked only once, DS-Flows benefit from continuous execution, where data is consumed continuously and each actor is invoked repeatedly, as upstream actors produce new results or real-world observations become available.

Definition 5 (Execution Step of a DS-Actor). Let $G(V, E, l)$ be a DS-Flow, let $v \in V$ be a DS-Actor, and let $\mathbb{M} = \langle F, \mathcal{P}, \mathcal{D}, \mathcal{O}, \omega_I, \omega_O, \Delta \rangle$ be the corresponding model. Each execution step s for DS-Actor v has a corresponding input and output window, $in(s)$ and $out(s)$

- if s is a source node in G, then $in(s) = \perp$ and $out(s) = [s \times \Delta, s \times \Delta + w_{out})$;
- if s is not a source node, then $in(s) = [s \times \Delta, s \times \Delta + w_{in})$ and $out(s) = [s \times \Delta, s \times \Delta + w_{out})$, where Δ is the value of the shift parameter and w_{in} and w_{out} are the input and output window sizes defined by ω_I and ω_O, respectively. ◇



(a) MDG with three models (b) MDG with five models

Fig. 4. Two sample *model dependency graphs* (MDG) with different numbers of constituent models.

For example, a hurricane model generates rainfall data, which will intuitively have an effect on the output of a flooding model. An example of these dependencies is visualized as a graph in Figure 4. In this example, the hurricane model is the initial model; it does not take any model-produced inputs, as there are very few inputs that can influence the path of a hurricane. This model starts with a *control token*. The dependency graph is then traversed in a breadth-first fashion so descendant models' input windows can be satisfied.

The graph describing the workflow is traversed *asynchronously*. When traversing a node, the ownership of the control token is not deterministic, but is based instead on the progress towards the goal that the model at each node has made. It may be passed if the model has completed the simulation and reported its results, or it may be retained at a particular state to allow for additional processing to occur. When a single traversal has been completed, another traversal begins immediately to provide additional opportunities for control token resolution to occur. As we describe below, the execution of each DS-Actor node is thus non-atomic: Each DS-Actor itself has a subgraph of processing modules responsible for different phases of its execution.

3.4 DataStorm Actors (DS-Actors)

The core component of the `DataStorm-FE` system is the `DataStorm`-Actor or DS-Actor, consisting of several more specialized sub-modules that adapt the domain simulator to interface with the wider system. Each DS-Actor in a `DataStorm` workflow is associated with one DS-Model. The role that the DS-Actor submodules play are summarized as follows:

- **Data Pre-processing:** The relevant data from upstream DS-Actors and external sources are ingested and aligned before used for simulations.
  - **Window Manager** handles temporal window alignment between models, ensuring sufficient data exist for input window satisfaction, repackaging upstream results into temporally aligned subsets.
  - **Alignment Manager** receives the subsets from the Window Manager, converts the raw results into a format the running model can process, and handles spatial realignment. It reduces or increases the resolution of the inputs (if necessary) to ensure spatial and temporal alignment between input data resolution and model requirements.
- **Ensemble Creation:**
  - **Sampling Manager** samples the possible simulation space, balancing computation budget against potential scenario fan-out, to decide which simulation scenarios to execute. It considers previous results, user inputs, computational budgets, and parametric inputs to generate a high-dimensional simulation space, representing all possible simulation configurations that could be executed. Additionally, the simulation budget may take into account metadata associated with the computational costs of each model, providing a wider fan-out for cheaper simulations and more aggressive pruning for expensive ones.
  - **Execution Manager** executes the simulator configurations marked for ensemble inclusion within the cluster, allocating and load balancing instances as necessary. It provides

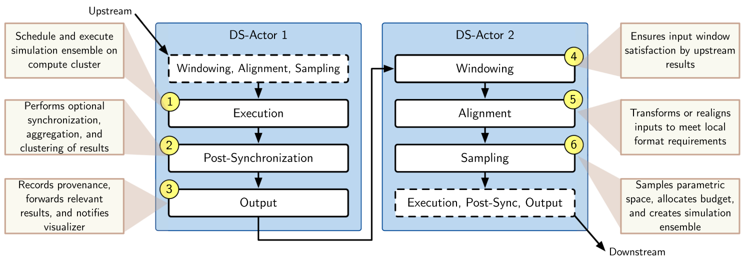

Upstream
Schedule and execute simulation ensemble on compute cluster
Performs optional synchronization, aggregation, and clustering of results
Records provenance, forwards relevant results, and notifies visualizer
DS-Actor 1
Windowing, Alignment, Sampling
1
Execution
2
Post-Synchronization
3
Output
DS-Actor 2
4
Windowing
5
Alignment
6
Sampling
Execution, Post-Sync, Output
Ensures input window satisfaction by upstream results
Transforms or realigns inputs to meet local format requirements
Samples parametric space, allocates budget, and creates simulation ensemble
Downstream

Fig. 5. A detailed view of DS-Actors with individual sub-modules, numbered by order of discussion.

a unified interface for running distributed, heterogeneous simulator implementations in parallel when possible and in sequence if not. DataStorm-FE provides an orchestration layer (using Ansible to automate configuration management and Vagrant for provisioning and deployment) for creating, configuring, and destroying instances as needed [6].

- **Post-processing:** Once the simulation instances have been executed, the resulting data must be collected or aggregated before being passed to the downstream DS-Actor(s).
  - **Post-Synchronization Manager** offers an opportunity to synchronize, aggregate, and transform the outputs of simulation instances in the ensemble. It aggregates or prunes raw results or increases predictive accuracy through result post-processing.
  - **Output Manager** packages and stores the results in the provenance database before passing data to the next downstream DS-Actor. It is also responsible for notifying the visualization server that new results are ready to be displayed.

Consider the following DS-Flow scenario: The user selects several possible hurricane scenarios, which correspond to distinct inputs to $\mathbb{M}_{hurricane}$. Multiple simulator instances spin up and evaluate the proposed scenarios, producing different rainfall maps, wind speeds, and hurricane tracks, for a given spatio-temporal frame. These outputs need to be aggregated and passed to the downstream actors. These are spatiotemporally aligned and converted to match the expected inputs to the flooding and mobility models. For $\mathbb{M}_{flood}$, the Sampling Manager chooses a subset of possible flood scenarios to be executed and these scenarios are pushed to the Execution Manager to run the corresponding flood simulation instances. $\mathbb{M}_{mobility}$ waits until its $I(s)$ is satisfied, i.e., the outputs from both $\mathbb{M}_{hurricane}$ and $\mathbb{M}_{flood}$ are ready. Once the data are pre-processed, the alignment, conversion, and sampling processes repeat for $\mathbb{M}_{mobility}$ to generate the corresponding simulation ensemble.

In the rest of this section, we detail the above sub-modules of DataStorm actors further. As we see in Figure 5, we start the discussion of the sub-modules from the Execution Manager.

3.4.1 Execution Manager.

The main role of the DS-Actor is to execute simulations, based on the current data, within the user-specified parameter space: As briefly summarized above and further formalized below, the DS-Actor samples vectors, $\vec{p}$, from the parameter space, $\mathcal{P}$, to create simulation instances to execute (we discuss the Sampling Manager in further detail in Section 3.4.6).

Definition 6 (Simulation Instance). A simulation instance, $\mathbb{S}(\vec{p})$, is a tuple $\langle F, \vec{p}, \mathcal{D}, O \rangle$, where the set, $\mathcal{P}$, of simulation parameters has been instantiated as $\vec{p}$. ◇

Each simulation execution instance is assigned to a compute resource (see Figure 8) to execute $F(\vec{p}, \vec{d})$ for a given input data vector, $\vec{d}$; the resulting output vector, $\vec{o}$, is returned back to DataStorm.

Definition 7 (Execution Instance). A model execution instance, $\mathbb{E}(\vec{p}, \vec{d})$, is a tuple $\langle F, \vec{p}, \vec{d}, O \rangle$, where simulation parameters and input data have been instantiated to $\vec{p}$ and $\vec{d}$, respectively; each execution instance also has a corresponding output data instance $\vec{o}$. ◇

Definition 8 (Execution Manager). Let $\mathbb{M}$ be a model. At each simulation step, s, the Execution Manager takes as input n parameter and input instantiation pairs, $\mathfrak{I}_{(s)} = \{(\vec{p}_{(s),1}, \vec{d}_{(s),1}), \ldots, (\vec{p}_{(s),n}, \vec{d}_{(s),n})\}$ which we refer to as *simulation jobs*, and returns n output vectors $\mathfrak{O}_{(s)} = \{\vec{o}_{(s),1}, \ldots, \vec{o}_{(s),n}\}$, each resulting from the processing of a simulation job by a corresponding model instantiation. We refer to this collection of instances as a *simulation ensemble.* ◇

When executing a simulator, there may be potentially many plausible scenarios to explore, each representing one possible way that future events play out; we call these alternate timelines *causality traces.* These traces may branch, representing the exploration of multiple scenarios; they may be pruned, representing a loss of interest or confidence in that scenario; or they may merge, representing the rejoining or aggregation of multiple compatible (see Section 3.4.4) traces back into one. A special provenance data structure is used to keep track of these traces.

Definition 9 (Execution Provenance). We also define a special provenance variable, π, such that $\vec{o}_{(s),j}[\pi] = (\mathbb{M}, s, \vec{p}_{(s),j}, \vec{d}_{(s),j}) \odot \vec{d}_{(s,j)}[\pi]$, where $\odot$ denotes the concatenation operation; intuitively, the variable contains the accumulated operations that contributed to a particular output. ◇

Internally, the data provenance is a tree-like directed acyclic graph. In this provenance graph, the root is a particular outcome and the tree collectively encodes the parameters, assumptions, and predictions that led to that outcome.

3.4.2 Post-synchronizer. Simulation results passed by one DS-Actor to the other may be an aggregate of multiple execution instances in the ensemble. It is the responsibility of the Post-synchronizer sub-module to aggregate simulation outputs as appropriate:

Definition 10 (Post-synchronizer). Let $\mathbb{M}$ be a model and let $\mathfrak{O}_{(s)} = \{\vec{o}_{(s),1}, \ldots, \vec{o}_{(s),n}\}$ be n output triples returned by the Execution Manager at step s. Given a target number of outputs, m, an aggregation function, $A()$, an aggregation criterion, θ_A, and a compatibility criterion, θ_C, the Post-synchronizer creates a new set $\mathfrak{R}_{(s)} = \{\vec{r}_{(s),1}, \ldots, \vec{r}_{(s),m}\}$, where $\vec{r}_{(s),j} = A(O_j)$ is the aggregation of the data in $O_j = \{o_{j,1}, \ldots\} \subseteq \mathfrak{O}_{(s)}$, selected based on the aggregation criterion, consisting of compatible data elements in $\mathfrak{O}_{(s)}$. ◇

In `DataStorm` terminology, we refer to each $\mathfrak{R}_{(s)}$ as a **`DataStorm` Aggregate Record (DSAR)** and each $\vec{r}_{(s),1} \in \mathfrak{R}_{(s)}$ as a **`DataStorm` Instance Record (DSIR)**; these terms are described below. DSIRs are packaged as a DSAR and placed in the DS-Actor output buffer by the Output Manager.

Note that the aggregation and compatibility criteria consider both data and provenance variables to decide whether two execution instances of a given model can be combined into one instance. Intuitively, depending on the application, two execution instances can be said to be compatible if their execution provenances (including the parameters used for the simulations) are similar.

The user constructs compatibility heuristics using numeric operators to encode potentially domain-specific constraints between interrelated model parameters. For instance, simulations with snowfall might be incompatible with results where temperatures significantly exceed 0°C, or hurricane models with different Saffir–Simpson categories, a domain-specific quantization, might be incompatible.

The Post-synchronizer then uses these heuristics to generate a record compatibility graph based on the simulation parameters of the execution provenance. This graph is used to perform a spectral clustering of the results, tuned via user-specified parameters, to aggregate compatible records.

Once execution instances are aggregated, the system also creates an aggregation provenance to keep track of the corresponding aggregation operations.

Definition 11 (Aggregation Provenance). The provenance variable associated with the aggregated results are computed as $\vec{r}_{(s,j)}[\pi] = (score_{agg}, score_{comp}, \{\langle o_{j,1}, contr_{j,1}\rangle, \ldots\}) \odot \vec{d}_{(s)}[\pi]$, where $score_{agg}$ is a measure of aggregation quality, $score_{comp}$ is the overall compatibility score, and $contr_{j,h}$ is a measure of the contribution of $o_{j,h} \in O_j$ to $\vec{r}_{(s,j)}$. ◇

3.4.3 Output Manager. Data within DataStorm is stored in an intermediate format prior to being shared within the system by the Output Manager. The data structures used by DataStorm are divided into three levels of abstraction: DSARs, DSIRs, and **DataStorm Fragmentary Records (DSFRs)**:

- DSFRs represent the most-granular form of information, storing raw results at a given spatiotemporal point. The resolution of these points vary according to the output resolution of the model that generated them. These results are used whenever it is required to operate directly on the data itself, such as during aggregation, clustering, or display.
- DSIRs encapsulate a collection of DSFRs and add metadata about the collective spatiotemporal context of the results. This type of record directly maps to a single execution of a simulator and includes information about the parametric configuration of the model that generated those results. In cases where one or more DSIRs are aggregated, a new DSIR (and associated DSFR(s)) is synthesized, ensuring the preservation of the constituent results. Note that DSIRs contain references to raw output data, spatial and temporal context for that data, and other metadata required for replication.
- DSARs are the highest level of abstraction, and intuitively represent a collection of one or more DSIRs, with a spatiotemporal context equal to the union of its constituent DSIRs. DSARs contain a broader set of metadata about the contained information, as well as including information about the provenance of that data, including of the ancestors (upstream inputs) to the DS-Actor that produced it.

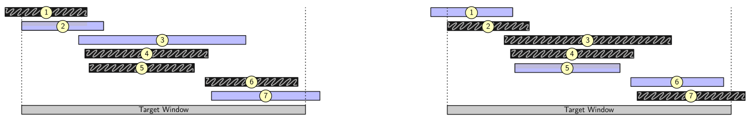
3.4.4 Window Manager. The data that is passed into the downstream actors by the Output Manager are received by the *input buffers* of these actors:

Definition 12 (Input Buffers). Let $G(V, E, l)$ be a DataStorm workflow, let $v_i \in V$ be a DS-Actor, and let $\mathbb{M}_i = \langle F_i, \mathcal{P}_i, \mathcal{D}_i, \mathcal{O}_i, \omega_{I_i}, \omega_{O_i}, \Delta_i\rangle$ be its corresponding model.

For each edge $(v_j \rightarrow v_i) \in E$, the actor v_i has access to a buffer, $B_{j,i}$, consisting of the sequence, $B_{j,i} = \{R_{j,(0)}, R_{j,(1)}, R_{j,(2)}, \ldots\}$ of DSARs produced by model $\mathbb{M}_j$ associated with DS-Actor v_j. ◇

When a model receives a set of DSARs from its upstream models into its input buffers, it must first determine how to map these results to the information required for its execution. This might be as simple as making sure necessary data is available for the temporal period being simulated or as complex as mixing and matching large numbers of inputs from many alternative scenarios to best match the experimental environment. This process is handled by the Window Manager, which produces one or more sets of DSARs. Each set of DSARs is referred to as a *windowing set*, with the DSARs encapsulated therein satisfying the input requirements ($I(s)$) of the model. Intuitively, the raw inputs can be assembled in many possible configurations, reflected as their power set. From this power set, windowing sets are selected to be used as inputs to the current step of the DS-Actor.

Definition 13 (Windowing Set). Let $G(V, E, l)$ be a DataStorm workflow, let $v_i \in V$ be a non-source DS-Actor, and let $\mathbb{M}_i$ be the corresponding model. As defined in Definition 5, for the execution of step, s, the DS-Actor v_i, has a corresponding input window, $in_i(s) = [s \times \Delta_i, s \times \Delta_i + w_{in_i})$.



(a) Least gap: {2, 3, 7} (b) Least overlap: {1, 5, 6}

Fig. 6. Two sample windowing strategies: The window manager receives seven data entries from the upstream actor and needs to combine these in a way that satisfies the windowing constraint (in this example, we assume that all data entries are compatible with each other).

Given a target number, k, of candidates, a windowing criterion, θ_W, and a compatibility criterion, θ_C, for each step s, the Window Manager of the DS-Actor v_i creates (up to) k windowing sets, $\mathfrak{W}_{(s),1}, \ldots, \mathfrak{W}_{(s),k}$. Each windowing set, $\mathfrak{W}_{(s),h}$, consists of inputs from the upstream DS-Actors:

$$\mathfrak{W}_{(s),h} = \{W_{(s),h,j} \mid (v_j \rightarrow v_i) \in E\}$$

where

- $W_{(s),h,j} = \{\vec{d} \mid (\vec{d} \in R_{j,(l)}) \wedge (R_{j,(l)} \in B_{j,i})\}$ such that
  - $W_{(s),h,j}$ satisfies the windowing criterion $\theta_W()$, indicating that the output scopes corresponding to $R_{j,(l)}$ collectively cover the input scope $in_i(s)$; and
  - $\mathfrak{W}_{(s),h}$ satisfies the compatibility criterion $\theta_C()$, indicating that the input vectors that contribute to all $W_{(s),h,j} \in \mathfrak{W}_{(s),h}$ are all compatible with each other. ◇

Windowing criteria help determine which subsets of data will be used as input for the given window based on data overlaps and data gaps. Consider the scenarios in Figure 6:

(a) The *least gap* strategy *minimizes gaps*. Let W be a set of data entries from upstream nodes and *window* be a window of interest. Let also $covered(W, window)$ be the portion of the *window* covered by the data in W. Under this strategy, we maximize $score_{win;lg}(W, window) = covered(W, window)$, In Figure 6(a), the algorithm selects input data scopes 2, 3, and 7, producing no gaps over the target window.

(b) The *least overlap* strategy *minimizes the overlaps while maximizing the total area of the window covered*. Under this criteria, we maximize $score_{win;lo}(W, window) = covered(W, window)/(1 + overlap(W, window))$, where $overlap(W, window)$ is the portion of the window covered by overlaps among W. In Figure 6(b), the algorithm selects input data 1, 5, and 6.

Once the windowing sets have been identified and selected, provenance information is recorded:

Definition 14 (Windowing Provenance). Each windowing set, $\mathfrak{W}_{(s),h}$, is associated with a provenance metadata,

$$\begin{aligned}
\mathfrak{P}_{(s),h} &= \langle score_{comp}, \{prov_{(j,i)} \mid (v_j \rightarrow v_i) \in E\}\rangle, \\
\textit{where } score_{comp} &= \textit{the overall compatibility score}, \\
prov_{(j,i)} &= \langle \mathbb{M}_i, score_{win,i}, \{\vec{d}[\pi] \mid (\vec{d} \in W_{(s),h,j}) \wedge (W_{(s),h,j} \in \mathfrak{W}_{(s),h})\}\rangle, \\
\textit{and } score_{win,i} &= \textit{the measure of windowing quality for each input buffer}.
\end{aligned}$$
◇

Thus, the k windowing sets, $\mathfrak{W}_{(s),1}, \ldots, \mathfrak{W}_{(s),k}$, and the corresponding provenance metadata are passed to the Alignment Manager.

3.4.5 Alignment Manager. Once windowing sets satisfying the running model's input requirements are identified, the corresponding data must be transformed into a format that can be directly used by the model itself. This process varies and may include upsampling or downsampling spatial or temporal resolutions, chunking or truncating over-large inputs, aggregating overlapping data, or interpolating for missing data.

Definition 15 (Aligned Vectors). Let $G(V, E, l)$ be a `DataStorm` workflow, let $v_i \in V$ be a non-source DS-Actor, and let $\mathbb{M}_i$ be the corresponding model.

Let $\mathfrak{W}_{(s),1}, \ldots, \mathfrak{W}_{(s),k}$ be the k windowing sets created by the Window Manager for this model in step s and let $\mathfrak{P}_{(s),1}, \ldots, \mathfrak{P}_{(s),k}$ be the corresponding provenance vectors.

Given an alignment function, $\mathcal{A}()$, the Alignment Manager aligns each $\mathfrak{W}_{(s),h}$ to create a data vector to be processed by the model. More specifically, the alignment process combines all $W_{(s),h,j} \in \mathfrak{W}_{(s),h}$ into a single aligned vector $\vec{a}_{(s),h}$, such that

- $\vec{a}_{(s),h} = \mathcal{A}(\mathfrak{W}_{(s),h})$ is a data vector defined on the input space $\mathcal{D}_i$ of model $\mathbb{M}_i$. ◇

It is important that, during this process, the original input data is not modified or deleted, since this could obfuscate important data provenance relationships. As such, the resulting transformed data is treated as a new entry in the DS-Flow, and information about the process (such as uncertainty caused by interpolation) is added to the provenance graph.

Definition 16 (Alignment Provenance). The provenance data corresponding to the alignment process is

$$\vec{a}_{(s),h}[\pi] = \langle score_{align,h} \rangle \odot \mathfrak{P}_{(s),h},$$

where the alignment score, $score_{align,h}$, indicates the amount of information lost during the alignment process, which includes sub- or super-resolution of the input data to align the data resolution with the input temporal resolution, $res_{in,i}$ of model, $\mathbb{M}_i$. ◇

The resulting k aligned vectors $\vec{a}_{(s),1}, \ldots \vec{a}_{(s),k}$ are then passed to the Sampling Manager to help select simulation instances to execute.

3.4.6 Sampling Manager. The Sampling Manager takes the k aligned input data vectors and creates an ensemble of execution instances to be executed by the Execution Manager:

Definition 17 (Simulation Sampling). Given k data vectors $\vec{d}_{(s),1}, \ldots \vec{d}_{(s),k}$ and a target number n of execution instances, the Sampling Manager samples the parameter space defined by $\mathcal{P}$ to create the set $\mathfrak{I}_{(s)} = \{(\vec{p}_{(s),1}, \vec{d}_{(s),1}), \ldots, (\vec{p}_{(s),n}, \vec{d}_{(s),n})\}$, such that

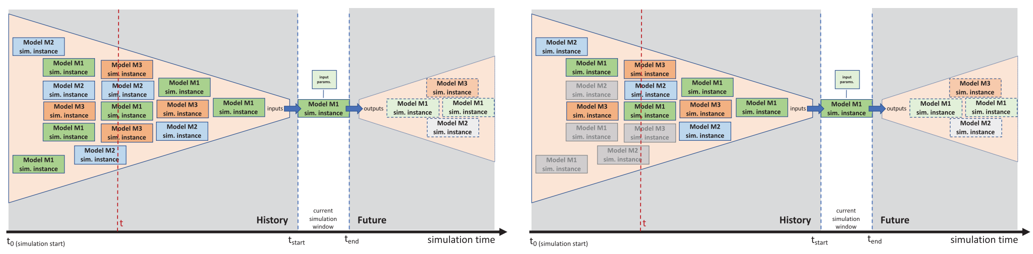
- the data vector $\vec{d}_{(s),h}$ corresponds to one of the aligned vectors in $\{\vec{a}_{(s),1}, \ldots \vec{a}_{(s),k}\}$,
- $\vec{d}_{(s),h} \in \mathcal{P}$,
- $\vec{d}_{(s),h}[\pi] = \langle score_{sample,h} \rangle \odot \vec{d}_{(s),h}[\pi]$, where $score_{sample,h}$ is the sampling quality score for this simulation sample. ◇

The resulting n input instantiation pairs (i.e., the simulation ensemble to be executed) are passed to the Execution Manager, as specified in Section 3.4.1.

Note that deciding which execution instances to include in the ensemble is far from trivial. We discuss this in greater detail in Section 4.

3.5 Timelines

As we described in the Introduction and the previous subsection, `DataStorm` supports *coupled* and *continuous* simulation workflows.



(a) History and future of a simulation instance (b) A possible timeline

Fig. 7. (a) Provenance captures the complete history that lead to the execution of a particular simulation instance; (b) a timeline, however, captures a maximal subset of an *execution provenance* such that for each given time instant, t, there is at most one simulation instance of each model type.

Each simulation instance in this workflow can be considered as a quadruple $\langle m, p, w, o \rangle$, indicating the model that has been executed (m), using input parameters and data (p), and generating output o corresponding to a time window $w = [t_{start}, t_{end}]$. Moreover, each instance has a *history* of other simulation instances (of the same or different model types) and the output of this simulation instance feeds into one or more *futures* that consist of simulation instances (of the same or different model types) that depend on the outcome of this particular instance.

In the previous subsection, we referred to the complete history of an executed simulation instance as an *execution provenance* and highlighted that the data provenance is a tree-like directed acyclic graph, where the root is a particular simulation instance and the tree collectively encodes the parameters, assumptions, and predictions that led to the execution of that particular instance (Figure 7(a)). Note that, ideally, in the simulation instances contained in the execution provenance, for each given time instant, t, there will be one and only one simulation instance of type m whose window w covers that time instant. However, due to the existence of fusion sub-actors that implement operations, such as *windowing*, *alignment*, and *aggregation*, each of which potentially combines simulation results from different execution instances, the provenance of a simulation instance does not necessarily satisfy this ideal scenario: Instead, each time instant t may be covered by *multiple* simulation instances of type m, each with a different set of data and parameter values.

We therefore define a *timeline* as a maximal subset of an *execution provenance*, such that for each given time instant, t, and for each model m, there is at most one simulation instance of that model type whose window w covers the time instant t (Figure 7(b)). Intuitively, each *timeline* corresponds to a different causal history and an *execution provenance* may contain many such histories, each of which can be separately displayed. The set is *maximal* in the sense that (a) the simulation time instants that are not covered by simulation instances of all models are minimized and (b) the causal relationships (encoded as input-output pairing among simulation instances) are maximally preserved (i.e., if a simulation instance is in the timeline, then other simulation instances that have produced data that are used as input by this simulation instance are also preferably in the timeline). Note that, since the number of possible timelines of a simulation execution can be large, `DataStorm` provides visualization actors that help explore these alternate timelines.

3.6 DataStorm Cloud Deployment and Configuration

Architecturally, `DataStorm` includes many moving parts, requiring substantial engineering automation to coordinate (Figure 8). The simulator instantiation and execution itself is managed by `StormCloud`, the cloud automation of `DataStorm`. It follows the concept of using multiple cloud

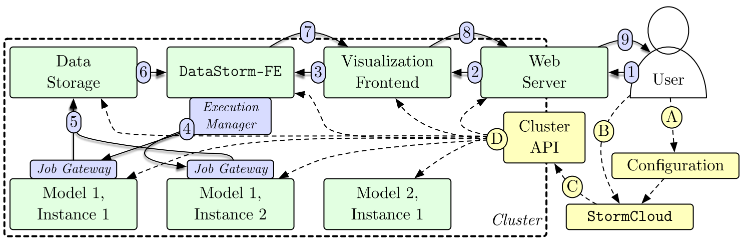


Fig. 8. DataStorm deployment (A–D, in yellow) and workflow execution (1–9, in blue).

providers simultaneously, sometimes known as "sky computing" [23], to leverage the complementary strengths of each. Specifically, our system uses Amazon Web Services' EC2, the Chameleon platform [22], an OpenStack-based provider, as well as Google Cloud Platform's Google App Engine, as part of a hybrid system. To facilitate the automatic creation and deletion of simulation instances, StormCloud integrates with relevant platform APIs, wrapping them in an abstraction layer, and allows for extension to alternative cloud providers.

The yellow boxes in Figure 8 represent the process through which a new cluster is configured for DataStorm deployment: The user (A) creates a configuration file using an interactive, guided tool to collect all relevant workflow parameters and settings. The user (B) then starts StormCloud, which reads this information and begins the process. It first (C) communicates with the cluster API, using it to (D) build, configure, and deploy the necessary infrastructure to reflect the user's described configuration. These tools are designed to be intuitive for non-technical domain experts.

Once DataStorm is deployed, it can run simulation workflows: A user (1) initiates an experiment through the web interface. This (2) triggers a call to the visualization frontend, which in turn (3) updates the core DataStorm-FE state machine. Once the previous sub-actors have produced a set of simulation jobs, the Execution Manager determines the appropriate load balancing for the current model, then (4) allocates them to the active instances. These jobs are translated from DataStorm's internal data format via a model-agnostic middleware, the Job Gateway. The models are then executed, and once the simulation results are ready, the same middleware (5) reformats the results and stores them in a harmonized format in the database. DataStorm-FE (6) reads the database to determine if the results are ready, and (7) notifies the visualization frontend they are ready for display. These are then (8) rendered to the browser for (9) further exploration by the user.

3.7 DataStorm Visualization

Results generated by a simulation ensemble are only as useful as the conclusions they produce. As the raw data generated by DataStorm can be both large and difficult to parse, and users may be domain experts and not necessarily data scientists, it is critical that data exploration and analysis be as accessible as possible. DataStorm provides an intuitive and easy-to-use interface that allows users to explore results in real time, as they are generated, from the familiar environment of their web browser. To best balance analytical power against usability, DataStorm provides a variety of tools to users, allowing deeper data comparisons without overloading users with overly complex default views.

We provide additional information about DataStorm visualization, with focus on timeline exploration, in the attached Appendix (supplementary material).

3.7.1 DataStorm Visualization Actors (DS-VizActors). Results from a heterogeneous simulation ensemble can be highly diverse in their modalities, resolutions, quantity, and more. Users must be able to explore rich and potentially voluminous data flexibly to allow both high-level analysis about cross-domain interactions and also highly focused assessments of domain-specific results by experts. DataStorm offers this functionality through a special type of DS-Actor referred to as **DataStorm visualization actors (DS-VizActors)**. DS-VizActors consist of a Window Manager, to receive data from the DS-Actors in the system, and an Alignment Manager, to align the resolutions of the input data from multiple models.

Once the data from different models are aligned, each selected data variable is then mapped into a visualization layer, represented by a suitable visualization paradigm, such as a heat map, vector plot, point cloud, or Lagrangian density field. Users can filter results by model, variable, or modality, assigning each result to a separate layer in composite views that allow spatial and temporal exploration. Additionally, users can view up to four such compositions at once, each corresponding to a different timeline, allowing for side-by-side comparisons between results with different spatiotemporal contexts, compositions, or causality traces.

3.7.2 Provenance Visualization. A DataStorm ensemble can produce potentially many different possible scenarios from the inputs, parameters, and ground truth data it receives. In comparing these conclusions, and in particular interpreting their relative likelihoods and confidence values computed by DataStorm, a user may wish to explore the result provenance graph directly. By doing so, important decision points and sources of uncertainty can be identified, reported, or manually pruned or adjusted to take into account user expertise not otherwise encoded into the simulations themselves. For more information, please refer to Section 1.2 of the Appendix.

Users may also choose to explore and compare different causality traces (i.e., paths through the provenance graph) to directly evaluate the downstream differences resulting from a particular variable or uncertainty point. By evaluating these downstream results from the perspective of domain experts, potential future scenarios can be pruned or emphasized based on potential impact, plausibility, or other difficult-to-quantify heuristics.

3.7.3 Collaboration. Urban planning, disaster response, and other problems that can benefit from the predictive capabilities of DataStorm often require large-scale collaboration between subject matter experts across disparate domains. To facilitate this process, the DataStorm Visualizer includes support for collaboration lobbies, allowing users to synchronize their visualizations across many different clients. This optional functionality builds on top of rich independent data exploration capabilities to allow users to share insights, identify cross-domain data dependencies, and present conclusions to decision-makers without requiring data export, report generation, or anything more complex than a web browser.

4 PARTITION-STITCH SIMULATION SAMPLING FOR DENSITY-BOOSTED ENSEMBLE CREATION

As described in Section 3.4.6, simulation space sampling, to decide which simulation instances to execute to support decision-making, is a key task of the DataStorm system. Yet, as we have also discussed in Section 2.3, simulation ensembles, represented as high-dimensional arrays (or tensors), are inherently sparse due to the large volume of potential simulations to execute and the cost of executing each individual simulation instance:

- **Limited ensemble simulation budgets:** Since complex, inter-dependent parameters affected by complex dynamic processes have to be taken into account, *execution of simulation ensembles can be very costly*. This leads to *simulation budget constraints* that limit the number of simulations one can include in an ensemble.
- **Inherent data sparsity of simulation ensembles:** *A simulation ensemble is inherently sparse (relative to the space of potential simulations one could run)*, which constitutes a

significant problem in simulation-based decision-making. This leads to the following critical question: "*Given a parameter space and a fixed simulation budget, which simulation instances we should include in the ensemble?*"

Note that, despite the inherent sparsity of the simulation ensembles, because of the complexities of key processes and the varying scales at which they operate, experts often lack the means to drive conclusions from these ensembles. This leads to the need for *data analytics on simulation ensembles to discover broad, actionable patterns*. In this section, we describe a tensor-based framework to represent, create, and analyze large simulation ensembles.[1]

4.1 Tensor-based Representation and Analysis of Simulation Ensembles

The tensor model maps a multi-attribute schema into an N-modal array. More formally, let I^j denote the number of distinct values that the jth attribute (or jth mode) can take. The tensor $\boldsymbol{\mathcal{X}}$ is then an N mode array such that $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. Intuitively, the modes of the tensor represent different *factors* that impact an observation and the value that the tensor records for a given cell corresponds to an observation for a specific combination of factor instances.

4.1.1 Tensor Representation of a Complex Dynamic System. Let us be given a complex dynamic system, S, with N input parameters, such that the ith input parameter can take I_i distinct values. For simplicity of the discussion, let us further assume that for each input parameter combination $\langle v_1, \ldots v_N \rangle$, the complex dynamic system S generates a single value $S(v_1, \ldots, v_n)$. Let, further, Y be the set of all simulations of the system S one can execute and the corresponding results; i.e., $Y = \{y_i = \langle\langle v_{i,1}, \ldots, v_{i,N}\rangle, S(v_{i,1}, \ldots, v_{i,N})\rangle \odot 1 \leq i \leq I_1 \times I_2 \times \cdots \times I_N\}$. It is easy to see that Y can be encoded as a tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, where for all $y_i \in Y$, the tensor cell $\boldsymbol{\mathcal{Y}}(v_{i,1}, \ldots, v_{i,N})$ has the value $S(v_{i,1}, \ldots, v_{i,n})$.

4.1.2 Tensor Representation of a Simulation Ensemble. Intuitively, the tensor model maps a multi-attribute schema to a multi-modal array (where each potential tuple is a tensor cell). Consequently, we can map a given simulation ensemble onto a tensor such that each simulation parameter corresponds to a mode of a tensor and the non-null entries in the tensor represent results of the simulations we have executed.

The number, $I_1 \times \cdots \times I_N$, of simulations of the system, S, one can run can be very large. Instead, as discussed in the introduction, we often run a much smaller subset (or ensemble) of the simulations to get an idea about S. Given an ensemble of $B \ll I_1 \times \cdots \times I_N$ simulations, let X be the set of simulations that have been selected to be executed as well as the corresponding system outputs; i.e., $X = \{x_i = \langle\langle v_{i,1}, \ldots, v_{i,N}\rangle, S(v_{i,1}, \ldots, v_{i,N})\rangle \odot 1 \leq i \leq B\}$. It is easy to see that X can be encoded as a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, where for all $x_i \in X$, the tensor cell $\boldsymbol{\mathcal{X}}(v_{i,1}, \ldots, v_{i,N})$ has the value $S(v_{i,1}, \ldots, v_{i,N})$ and all other cells have null values (indicating simulations that could potentially have been run, but have not been included in the ensemble). Since $B \ll I_1 \times I_2 \times \cdots \times I_N$, the tensor $\boldsymbol{\mathcal{X}}$ is very sparse, meaning that there will be many more null-valued cells than the cells recording real-valued simulation results.

4.1.3 Tensor-based Analysis of a Simulation Ensemble. Tensor decomposition [17, 28, 44] (which generalizes matrix decomposition to tensors) has been successfully used in various applications, such as social networks, sensor streams, and others [27]. Intuitively, the tensor decomposition process rewrites the given tensor in the form of a set of *factor* matrices (one for each mode of the input tensor) and a core matrix (which, intuitively, describes the spectral structure of the given tensor). As such, tensor decomposition has also been used for the analysis of dynamical systems:

[1]This is an extended version of the conference paper [29].

Reference [40] proposed a tensor-based model for time series and Reference [26] proposed a **dynamic mode decomposition (DMD)** scheme for the analysis of the behavior of complex dynamical systems.

The two most popular tensor decomposition algorithms are the Tucker [44] and the **CANDECOMP/PARAFAC (CP)** [17] decompositions. In this article, we focus on the Tucker decomposition of simulation ensemble tensors. Intuitively, the Tucker decomposition generalizes **singular value matrix decomposition (SVD)** to higher-dimensional data. Given a tensor $\mathcal{X}$, Tucker decomposition factorizes the tensor into factor matrices with different number of rows, which are referred to as the rank of the decomposition. For the simplicity of the discussion, let us consider a 3-mode tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$. Tucker decomposition would decompose $\mathcal{X}$ into three matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and one core dense tensor $\mathbf{g}$, such that

$$\mathcal{X} \approx \tilde{\mathcal{X}} = \mathbf{g} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \equiv \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_p \circ b_q \circ c_r,$$

where $\mathbf{A} \in \mathbb{R}^{I \times P}$, $\mathbf{B} \in \mathbb{R}^{J \times Q}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$, are the factor matrices and can be treated as the principal components in each mode. The (dense) core tensor, $\mathbf{g} \in \mathbb{R}^{P \times Q \times R}$, indicates the strength of interactions among different components of the factor matrices.

ALGORITHM 1: HOSVD

```
**Input**: Tensor $\mathcal{X}$, Rank for each mode $r_1, r_2, \ldots, r_N$
**Output**: Decomposed factors $U^{(1)}, U^{(2)}, \ldots, U^{(N)}$ and core tensor $\mathcal{G}$
**for** $n = 1, \ldots, N$ **do**
    matricize $\mathcal{X}$ into matrix $X_{(n)}$
    $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{(n)}$;
$\mathcal{G} = \mathcal{X} \times_1 U^{(1)T} \times_2 U^{(2)T}, \ldots, \times_N U^{(N)T}$
**return** $\mathcal{G}, U^{(1)}, U^{(2)}, \ldots, U^{(N)}$
```

More generally, given an N-mode tensor, $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, and N target rank values, r_1 through r_N, the corresponding Tucker decomposition is $[\mathcal{G}, \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}, \ldots, \mathbf{U}^{(N)}]$, such that

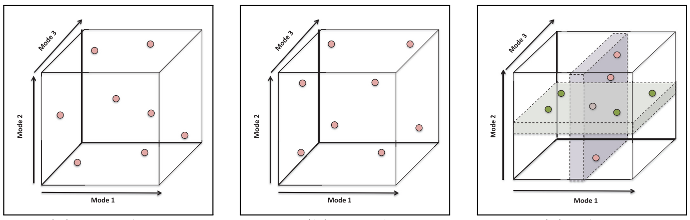
$$\tilde{\mathcal{X}} = \mathcal{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{U}^{(3)} \cdots \times_N \mathbf{U}^{(N)} \approx \mathcal{X}.$$

Here, $\mathbf{U}^{(i)}$ are the N factor matrices and $\mathcal{G}$ is an $r_1 \times \cdots \times r_N$ dimensional core tensor. Algorithm 1 illustrates the HOSVD algorithm [27] for Tucker decomposition of a given N-mode tensor, $\mathcal{X}$ for target rank values, r_1 through r_N. For each mode of the tensor, HOSVD matricizes (flattens) the high-order tensor $\mathcal{X}$ into a matrix. Then this matrix is decomposed (using SVD) to obtain the left eigenvectors and these are packed into a factor matrix for the corresponding mode. Finally, the core tensor is recovered from the original tensor and the N factor matrices obtained as described.

It is important to note that tensors very rarely have exact Tucker decompositions. In almost all cases, the new tensor $\tilde{\mathcal{X}}$ obtained by recomposing the factor matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and core tensor $\mathbf{g}$ is often different from the input tensor, $\mathcal{X}$. The accuracy of the decomposition is often measured by considering the Frobenius norm of the difference tensor. A major challenge, however, is that the sparser the input tensor, the less accurate will be its analysis—this is especially a major challenge in the analysis of inherently sparse simulation ensembles.

4.1.4 Simulation Space Sampling for Effective Ensemble Analysis. Ideally, to study the system, S, we would construct a complete tensor $\mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, and given target rank values, r_1 through r_N, we would obtain its corresponding Tucker decomposition $[\mathcal{H}, \mathbf{V}^{(1)}, \mathbf{V}^{(2)}, \mathbf{V}^{(3)}, \ldots, \mathbf{V}^{(N)}]$, where

$$\tilde{\mathcal{Y}} = \mathcal{H} \times_1 \mathbf{V}^{(1)} \times_2 \mathbf{V}^{(2)} \times_3 \mathbf{V}^{(3)} \cdots \times_N \mathbf{V}^{(N)} \approx \mathcal{Y}.$$



(a) Random (b) Grid (c) Slice

Fig. 9. Conventional solutions for ensemble generation.

However, this would be prohibitively costly: First, this would require $I_1 \times \cdots \times I_N$ simulations, which can be computationally overwhelming. Even if this many simulations can be obtained, the analysis of the resulting tensor may be prohibitively expensive. Instead, given a budget $B \ll I_1 \times \cdots \times I_N$, the problem is to identify a set, $X = \{x_i = \langle\langle v_{i,1}, \ldots, v_{i,N}\rangle, S(v_{i,1}, \ldots, v_{i,N})\rangle \odot 1 \leq i \leq B\}$ of B simulations to execute, such that the Tucker decomposition $[\mathcal{G}, \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}, \ldots \mathbf{U}^{(\mathrm{N})}]$ of the corresponding tensor $\boldsymbol{\mathcal{X}}$ has the following property:

$$\tilde{\boldsymbol{\mathcal{X}}} = \mathcal{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{U}^{(3)} \cdots \times_N \mathbf{U}^{(\mathrm{N})} \approx \boldsymbol{\mathcal{X}},$$

and the Frobenius norm, $\|\boldsymbol{\mathcal{Y}} - \tilde{\boldsymbol{\mathcal{X}}}\|_F$, of the difference (from the complete ensemble, $\boldsymbol{\mathcal{Y}}$) is small.

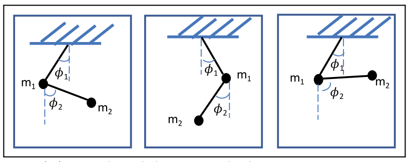
4.2 Key Observation: Density Boosting through Space Partitioning

Consider the three conventional ensemble sampling strategies visualized in Figure 9:

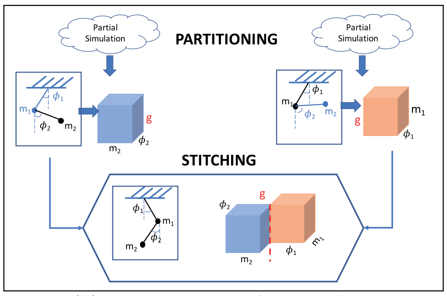
- *Strategy #1: Random Sampling*. The first approach for creating a budget constrained ensemble of simulations for the system S is to uniformly randomly sample $B \ll I_1 \times \cdots \times I_N$ parameter value configurations in the parameter space and execute those B randomly sampled simulations to obtain the ensemble, X_{rs} (Figure 9(a)).
- *Strategy #2: Grid Sampling*. The second approach for creating a budgeted ensemble for S is to sample B parameter value configurations at positions defined by a regularly spaced grid and execute those B sampled simulations to obtain the ensemble, X_{gs} (Figure 9(b)).
- *Strategy #3: Slice Sampling*. As we can see from Figures 9(a) and (b), the major difference between random sampling and grid sampling is that in grid-based ensemble construction, the subsets of the selected simulation samples are aligned on vertical and horizontal directions (or slices) of the underlying tensor, and these vertical and horizontal slices cover the tensor regularly. Alternatively, these slices and the samples within each slice can be randomly selected. Intuitively, each slice fixes one of the parameters, therefore, the samples within each slice are denser (whereas the density of the overall tensor remains the same). We refer to the resulting ensemble as X_{ss}.

It is easy to see that each of these conventional sampling strategies will lead to only sparse ensembles. These strategies cover the underlying parameter space in different ways using the same number of simulation instances. Consequently, while the local sub-space densities may differ, the overall simulation density is identically low for all three cases.

In this section, we describe a novel ensemble creation strategy, which we refer to as the *partition-stitch sampling*: given an N-parameter simulation and an ensemble budget of B, instead of randomly allocating the B samples in the N-dimensional parameter space, we partition the simulation space into $\sim N/2$ dimensional sub-spaces and allocate $B/2$ simulations for each sub-space: Note that, since the number of possible simulations for each sub-space reduced exponentially (in the



(a) A double pendulum system



(b) partition-stitch sampling

Fig. 10. Partition-stitch sampling for a five-parameter double pendulum system: (a) the initial angle of the first pendulum ϕ_1, (b) the initial angle of the second pendulum ϕ_2, (c) the weight of the first bob m_1, (d) the weight of the second bob m_2, and (e) the gravity, g.

number of excluded parameters), this corresponds to an exponential increase in the density of the samples for each sub-space:

Example 1 (Density Boosting through Space Partitioning). Let us consider a five-parameter *double equal-length pendulum* system in Figure 10(a). Instead of considering the original system, we can divide the simulation space into simulations for two three-parameter systems as we see in Figure 10:

- **System 1:** In this system, we are allowed to *vary* the initial angle, ϕ_1, and weight, m_1, of the first pendulum as well as the gravity, g; but the initial angle, ϕ_2, and weight, m_2, of the second pendulum are *fixed*.
- **System 2:** In the second system, we can *vary* the initial angle, ϕ_2, and weight, m_2, of the second pendulum as well as the gravity, g; in this case, the initial angle, ϕ_1, and weight, m_1, of the first pendulum are *fixed*.

Note that neither of the two systems are perfect representations of the overall behavior of the whole system as, in both cases, two out of the five parameters are fixed to some default values. However, the simulation densities of both systems are now much higher than the simulation density of the original system: Using the numbers considered earlier, each sub-system has three parameters with 20 distinct values, leading to a parameter space of $20^3 = 8{,}000$ simulations. If we allocate 500 (=1,000/2) simulations to each sub-space, this leads to a simulation density of $500/8{,}000 = 0.0625$, which is 200 time denser than the original simulation space. ◇

4.3 Key Challenge: Stitching Sub-spaces

While sub-space partitioning promises to boost densities of spaces, there remain several important questions to leverage this observation in practice. The first such critical question being

- "*How do we stitch back the results obtained from the individual sub-spaces?*"

Here, we may have several alternatives: In the simplest alternative, all the simulations from the two systems can be unioned into a single five-mode tensor and this five-mode tensor can be decomposed for analysis. This is potentially very expensive, as the decomposition cost often increases exponentially with the number of modes of the input tensor [25, 30]. We will also see that, once unioned into a single tensor, the overall density is still low and the accuracy gains will be very limited.

Instead, in this section, we will present a join-based scheme to increase the *effective density* of the ensemble. In particular, we will present two approaches (join stitching and zero-join stitching) to combine simulation results from the sub-systems and experimentally validate the effectiveness of these schemes. Several questions, however, remain:

- *How do we select the parameter to be shared across the two sub-spaces?*: We experimentally verify that the significant gains in accuracy due to the increase in simulation densities of the sub-systems reduces the need to be particularly careful in selecting the shared parameter.
- *What about the fact that both partial systems use some* default*values to fix some of the parameters? Does not this negatively affect accuracy?* We will see that the gains obtained in accuracy due to the significant jump in simulation densities will overcome any disadvantages associated with fixing some of the parameters.
- *If we are joining the sub-ensembles back to the original N-parameter space, would not this negatively effect the tensor decomposition cost?* If done naively, yes; and we discuss this in the rest of this section.

4.4 Partition-stitch Sampling

In this section, we show that, while executing the same number (B) of simulation instances as before, we can *increase the effective simulation density* of the ensemble by carefully partitioning the simulations to run into two groups and, then, by carefully stitching them, relying on *shared information* among these groups to *transfer* knowledge among them. The key observation is that most complex processes can be partitioned such that, while each partition captures different sub-processes, these nevertheless relate to each other and, hence, reflect the footprints of the same underlying global pattern. Therefore, at least in theory, it should be possible to partition the given system S into two sub-systems, S_1 and S_2, and analyze them independently. *Transferring* what we independently learned from the analysis of S_1 and S_2 back-and-forth, we should be able to gather information regarding the original global system, S. To leverage this observation, however, we need to answer two major questions: (a) "*How do we partition the system, S, into two sub-systems?*" and (b) "*How do we stitch the outcomes of these two sub-systems, S_1 and S_2, back to learn about S?*"

4.4.1 PF-Partitioning of a Parameter Space. It turns out that the answer to the first question is relatively straightforward: Given a system S with N input parameters, we will partition the system into two sub-systems S_1 and S_2, each with $\frac{N-k}{2} + k$ input parameters, such that

- the two systems share k of their input parameters as *pivot parameters*, and
- for each system, the remaining $\frac{N-k}{2}$ parameters will be set to a default value, referred to as *fixing constants*.

We will refer to this as the **Pivoted/Fixed (PF)**-*partitioning* of a parameter space. Intuitively, S_1 and S_2 correspond to two *constrained sub-spaces*: They have lesser free parameters than the original system S, as each one is generated by fixing $\frac{N-k}{2}$ of the input parameters. Once the two sub-systems are obtained through PF-partitioning, we can then create two sets, X_1 and X_2, of ensembles (through random, grid, or slice sampling), each with $B/2$ simulations— these simulations are created with common values for shared pivot parameters. Consequently, the pivot parameters can be used for stitching the two ensembles together. Formally, let $\rho_1, \ldots, \rho_i, \ldots \rho_N$ denote the N input parameters of S, each with a domain with I_i distinct values. Without loss of generality, we refer to

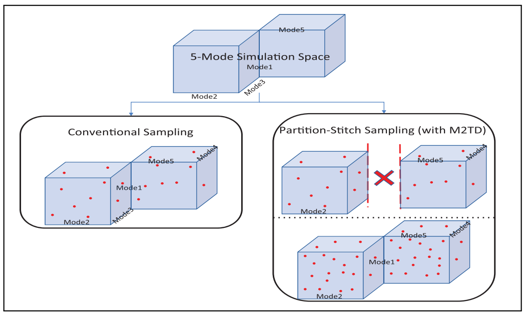


Fig. 11. Ensemble creation through PF-partitioning, followed by JE-stitching provides a higher effective density than the convention sampling of the original parameter space.

- ρ_1 through ρ_k as the pivot parameters,
  —we select $P \leq I_1 \times \cdots \times I_k$ possible configurations for the pivot parameters for ensemble generation,
- ρ_{k+1} through $\rho_{k+(N-k)/2}$ will serve as the free input parameters of system S_1 and fixed parameters of S_2,
  —we select $E \leq I_{k+1} \times \cdots \times I_{k+(N-k)/2}$ possible configurations for the free parameters for ensemble generation for system S_1,
- $\rho_{k+(N-k)/2+1}$ through ρ_N will serve as the free input parameters of system S_2 and fixed parameters of S_1,
  —we select $E \leq I_{k+(N-k)/2+1} \times \cdots \times I_N$ possible configurations for the free parameters for ensemble generation for system S_2.

Note that, given the input budget B, we have $P \times E = B/2$. In the next sub-section, we discuss how to stitch these sub-ensembles to increase the overall effective density.

4.4.2 JE-stitching. As we mentioned above, the goal of the stitching process is to increase the effective density of the ensemble. ***Join-Ensemble (JE)**-Stitching* achieves this by *joining* or *zero-joining* the two sub-systems along the shared modes:

Join-based Stitching. Let $\boldsymbol{\mathcal{X}}_1$ and $\boldsymbol{\mathcal{X}}_2$ denote the two tensors representing the simulation ensembles, X_1 and X_2, for the two sub-systems $S_1(\rho_{1,1}, \ldots, \rho_{1,k+(N-k)/2})$ and $S_2(\rho_{2,1}, \ldots, \rho_{2,k+(N-k)/2})$, respectively. For simplicity, let the first k parameters of both sub-systems denote the set of parameters shared between the two sub-systems. We construct a new join ensemble, J, as follows: For all pairs of simulations in the two ensembles that agree on the parameter values for the k shared parameters (i.e., $(\rho_{1,1} = \rho_{2,1}) \wedge \ldots \wedge (\rho_{1,k} = \rho_{2,k})$), we compute the average of the terms

$$x_1 = X_1(\rho_{1,1}, \ldots, \rho_{1,k}, \rho_{1,k+1}, \ldots, \rho_{1,k+(N-k)/2})$$

$$x_2 = X_2(\rho_{2,1}, \ldots, \rho_{2,k}, \rho_{2,k+1}, \ldots, \rho_{2,k+(N-k)/2})$$

and the resulting average, $\frac{x_1+x_2}{2}$, as the value for the corresponding join ensemble entry $J(\rho_{1,1}, \ldots, \rho_{1,k+(N-k)/2}, \rho_{2,k+1}, \ldots, \rho_{2,k+(N-k)/2})$.

Note that, since for each one of the P unique combinations selected for the shared pivot parameters, there are E ensemble simulations in both sub-systems, the resulting join ensemble tensor, $\mathcal{J}$, represents $P \times E^2$ joined simulations—since, as we saw in the previous subsection, we have $P \times E = B/2$, this gives us $B^2/(4P)$ simulation entries (and assuming that $B \gg (4P)$) *effectively squaring the simulation density* (Figure 11). As we experimentally verify in Section 5, (due to this increased effective density) the decomposition of $\mathcal{J}$ will be a far better approximation for the original system S than the decomposition of the tensor $\boldsymbol{\mathcal{X}}$, which represents the original set of simulations, $X = X_1 \cup X_2$. In fact, the accuracy gains associated with this density increase

- prevents any disadvantages associated with eliminating some of the free parameters, and
- leads to significant overall accuracy gains, even without precise *a priori* knowledge about parameters to use as pivot and/or values for fixing constants.

Zero-Join based Stitching. Note, however, that when E (i.e., sub-system densities) is small, the overall join ensemble density may still be too low to provide accurate analysis. In such a case, we can further boost the overall ensemble density by using *zero-join* (as opposed to simple join) to stitch the sub-ensembles: When constructing the join ensemble, J, for all pairs of simulations in the two sub-ensembles that agree on the parameter values for the k shared parameters (i.e., $(\rho_{1,1} = \rho_{2,1}) \wedge \ldots \wedge (\rho_{1,k} = \rho_{2,k})$), we still compute the average of the terms as described above. But, in this case, if there is a simulation instance,

$$x_1 = X_1(\rho_{1,1}, \ldots, \rho_{1,k}, \rho_{1,k+1}, \ldots, \rho_{1,k+(N-k)/2})$$

but the simulation instance

$$X_2(\rho_{1,1}, \ldots, \rho_{1,k}, \rho_{2,k+1}, \ldots, \rho_{2,k+(N-k)/2})$$

does not exist; then, we treat the *missing* simulation instance as if it exists with 0 value, and we construct the corresponding join ensemble entry $J(\rho_{1,1}, \ldots, \rho_{1,k+(N-k)/2}, \rho_{2,k+1}, \ldots, \rho_{2,k+(N-k)/2})$ with value $\frac{x_1+0}{2}$. We similarly handle simulation instances in X_2.

Note that zero-joining increases the effective density of the simulation ensemble to $2 \times (P \times E^2) \times E^2$, and as we experimentally verify in Section 5, it significantly boosts accuracy in cases where sub-ensemble simulation densities are too low for basic join-based stitching be effective.

4.5 Multi-task Tensor Decomposition (M2TD)

The difficulty with JE-stitching, of course, is that tensor $\mathcal{J}$ has almost double the number of modes as the tensors $\boldsymbol{\mathcal{X}}_1$ and $\boldsymbol{\mathcal{X}}_2$. Consequently, its decomposition is likely to be significantly more expensive than the decomposition of these two pre-join tensors. What remains to be shown is that we can, in fact, obtain the decomposition of $\mathcal{J}$ directly from the decompositions of $\boldsymbol{\mathcal{X}}_1$ and $\boldsymbol{\mathcal{X}}_2$. We discuss this in this section.

Let $\boldsymbol{\mathcal{X}}_1$ and $\boldsymbol{\mathcal{X}}_2$ be two sub-ensemble tensors corresponding to sub-systems constructed through PF-partitioning of an N-parameter system, S. Let J be the join ensemble and $\mathcal{J}$ be the corresponding join tensor one could obtain through JE-stitching. In this section, we introduce three alternative M2TD schemes to obtain the decomposition of $\mathcal{J}$ from the decompositions of $\boldsymbol{\mathcal{X}}_1$ and $\boldsymbol{\mathcal{X}}_2$.

4.5.1 M2TD-Average (M2TD-AVG). Remember from the earlier sections that both $\boldsymbol{\mathcal{X}}_1$ and $\boldsymbol{\mathcal{X}}_2$ are M-modal tensors, where $M = k + (N-k)/2$, and that the first k modes are shared. We modify the HOSVD algorithm, presented in Section 4.1.3, to obtain the proposed M2TD-AVG algorithm (Algorithm 2). Intuitively, M2TD-AVG takes the first k factor matrix pairs, $(U^{1(n)}, U^{2(n)})$, corresponding to the shared pivot tensors of the independently decomposed tensors, $\boldsymbol{\mathcal{X}}_1$ and $\boldsymbol{\mathcal{X}}_2$, and averages each pair to obtain a *common* factor matrix representing both tensors: Since factor matrices, $U^{1(n)}$ and $U^{2(n)}$, both map the domain of the corresponding factor to a vector space represented by r_n singular factors (sorted in decreasing order of significance), the operation $average(U^{1(n)}, U^{2(n)})$ essentially constructs a new vector space, where each element of the domain is represented by the average vector from the two input vector spaces (Figure 15(a)). Remaining factor matrices are then combined to obtain the core tensor, $\boldsymbol{\mathcal{G}}$ (see Figure 12). As we experimentally verify in Section 5, this leads to a better approximation of the original system than any of the naive ensemble sampling schemes.

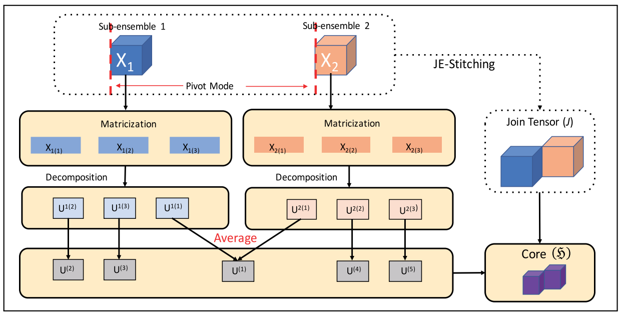


Fig. 12. Overview of M2TD-AVG.

ALGORITHM 2: M2TD-AVG

```
**Input**: Tensors $\mathcal{X}_1$ and $\mathcal{X}_2$, Rank for each mode $r_1, r_2, \ldots, r_N$
**Output**: Decomposed factors $U^{(1)}, U^{(2)}, \ldots, U^{(N)}$ and core tensor $\mathcal{G}$ for the join tensor $\mathcal{J}$
for $m = 1, \ldots, M$ do
    matricize $\mathcal{X}_1$ into matrix $X_{1(m)}$
    matricize $\mathcal{X}_2$ into matrix $X_{2(m)}$
for $n = 1, \ldots, k$ do
    $U^{1(n)} \leftarrow r_n$ leading left singular vectors of $X_{1(n)}$
    $U^{2(n)} \leftarrow r_n$ leading left singular vectors of $X_{2(n)}$
    $U^{(n)} \leftarrow average(U^{1(n)}, U^{2(n)})$
for $n = k + 1, \ldots, M$ do
    $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{1(n)}$
for $n = M + 1, \ldots, 2M - k$ do
    $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{2(n-M+k)}$
$\mathcal{J}$ = join_tensor($\mathcal{X}_1, \mathcal{X}_2$)
$\mathcal{G} = \mathcal{J} \times_1 U^{(1)T} \times_2 U^{(2)T}, \ldots, \times_N U^{(N)T}$
return $\mathcal{G}, U^{(1)}, U^{(2)}, \ldots, U^{(N)}$
```

4.5.2 M2TD-Concatenate (M2TD-CONCAT). M2TD-AVG, presented in the previous subsection, recovers the factor matrices for pivot parameters (modes) by averaging the corresponding factor matrices; i.e., by first obtaining the singular vectors of the matricizations and then averaging these singular vectors. However, there is nothing that guarantees that these averages will act as singular vectors themselves.

Instead, the alternative M2TD-CONCAT algorithm (detailed in Algorithm 3 and visualized in Figure 13) avoids this potential issue by first constructing a concatenated matricization for each pivot mode pair and then seeking the left singular vectors of this combined matricization. Intuitively, M2TD-CONCAT maps the matricizations along the shared/pivot modes back into the higher-modal space and seeks the singular vectors that best represent this higher modal space.

4.5.3 M2TD-Selection (M2TD-SELECT). The M2TD-CONCAT algorithm presented above tries to improve the vector averaging scheme of M2TD-AVG through row-by-row concatenation of the pivot matricizations before the corresponding factor matrices are computed. In this subsection, we note that there is an alternative, and potentially more effective, way to improve the M2TD-AVG scheme: Once the factor matrices for the pivots are obtained, instead of averaging them, we can carefully select between the individual rows of the corresponding factor matrices and use these selected rows to construct more effective combined factor matrices.

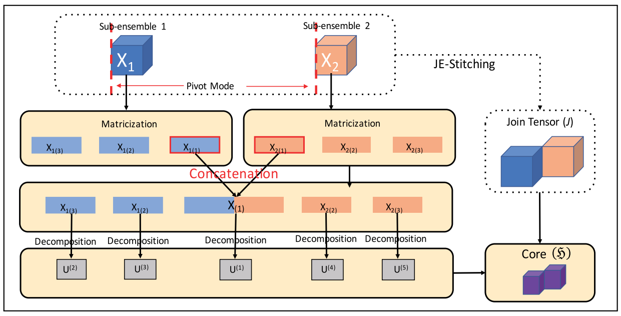


Fig. 13. Overview of M2TD-CONCAT.

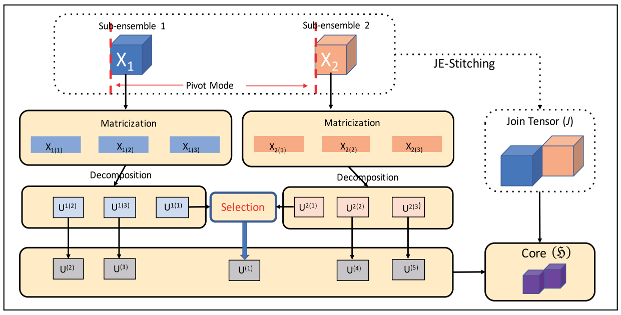


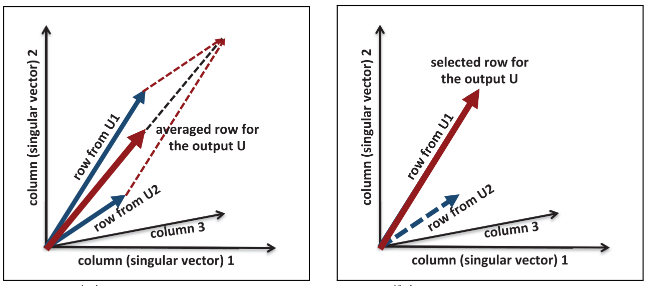
Fig. 14. Overview of M2TD-SELECT.

ALGORITHM 3: M2TD-CONCAT

```
Input: Tensors $\boldsymbol{\mathcal{X}}_1$ and $\boldsymbol{\mathcal{X}}_2$, Rank for each mode $r_1, r_2, \ldots, r_N$
Output: Decomposed factors $\boldsymbol{U}^{(1)}, \boldsymbol{U}^{(2)}, \ldots, \boldsymbol{U}^{(N)}$ and core tensor $\boldsymbol{\mathcal{G}}$ for the join tensor $\boldsymbol{\mathcal{J}}$
for $n = 1, \ldots, k$ do
    matricize $\boldsymbol{\mathcal{X}}_1$ into matrix $X_{1(n)}$
    matricize $\boldsymbol{\mathcal{X}}_2$ into matrix $X_{2(n)}$
    $X_{(n)} \leftarrow concatenate(X_{1(n)}, X_{2(n)})$
    $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{(n)}$
for $m = k + 1, \ldots, M$ do
    matricize $\boldsymbol{\mathcal{X}}_1$ into matrix $X_{1(m)}$
    matricize $\boldsymbol{\mathcal{X}}_2$ into matrix $X_{2(m)}$
for $n = k + 1, \ldots, M$ do
    $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{1(n)}$
for $n = M + 1, \ldots, 2M - k$ do
    $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{2(n-M+k)}$
$\boldsymbol{\mathcal{J}}$ = join_tensor($\boldsymbol{\mathcal{X}}_1, \boldsymbol{\mathcal{X}}_2$)
$\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{J}} \times_1 U^{(1)T} \times_2 U^{(2)T}, \ldots, \times_N U^{(N)T}$
return $\boldsymbol{\mathcal{G}}, U^{(1)}, U^{(2)}, \ldots, U^{(N)}$
```

The pseudocode for the process is shown in Algorithm 4 and visualized in Figure 14. Note that the major difference between this algorithm and M2TD-AVG is the line

$$U^{(n)} \leftarrow row\_select(U^{1(n)}, U^{2(n)}),$$


column (singular vector) 2
row from U1
averaged row for the output U
row from U2
column 3
column (singular vector) 1
selected row for the output U
row from U1
row from U2
column 3
column (singular vector) 1

(a) M2TD-AVG (b) M2TD-SELECT

Fig. 15. Comparison of the row construction processes between M2TD-AVG and M2TD-SELECT.

ALGORITHM 4: M2TD-SELECT

```
Input: Tensors $\mathcal{X}_1$ and $\mathcal{X}_2$, Rank for each mode $r_1, r_2, \ldots, r_N$
Output: Decomposed factors $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}$ and core tensor $\mathcal{G}$ for the join tensor $\mathcal{J}$
for $m = 1, \ldots, M$ do
    matricize $\mathcal{X}_1$ into matrix $X_{1(m)}$
    matricize $\mathcal{X}_2$ into matrix $X_{2(m)}$
for $n = 1, \ldots, k$ do
    $U^{1(n)} \leftarrow r_n$ leading left singular vectors of $X_{1(n)}$
    $U^{2(n)} \leftarrow r_n$ leading left singular vectors of $X_{2(n)}$
    $U^{(n)} \leftarrow row\_select(U^{1(n)}, U^{2(n)})$
for $n = k+1, \ldots, M$ do
    $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{1(n)}$
for $n = M+1, \ldots, 2M-k$ do
    $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{2(n-M+k)}$
$\mathcal{J}$ = join_tensor($\mathcal{X}_1, \mathcal{X}_2$)
$\mathcal{G} = \mathcal{J} \times_1 U^{(1)T} \times_2 U^{(2)T}, \ldots, \times_N U^{(N)T}$
return $\mathcal{G}, U^{(1)}, U^{(2)}, \ldots, U^{(N)}$
```

ALGORITHM 5: ROW_SELECT

```
Input: Factor matrices $U_1$ and $U_2$
Output: Row-selected Factor Matrix $U$
$I \leftarrow num\_rows(U_1)$
for $1 \leq i \leq I$ do
    if $\|row(U_1, i)\|_2 \geq \|row(U_2, i)\|_2$ then
        $row(U, i) \leftarrow row(U_1, i)$
    else
        $row(U, i) \leftarrow row(U_2, i)$
return $U$
```

where the factor matrix $U^{(n)}$ is constructed by selecting the appropriate rows from $U^{1(n)}$ or $U^{2(n)}$ instead of simply averaging them. This row selection process is further detailed in Algorithm 5 and visualized in Figure 15(b). As we see here, the key idea is to consider the energies (captured by the two-norm function) of each row, i, in U_1 and U_2, and identify which of the two factor matrices provides a higher energy for that particular row. Intuitively, this enables us to identify which of the two factor matrices better represents the entity corresponding to row, i, and, given this information, we can construct the row i of the output factor matrix, U by selecting the corresponding row from the factor matrix, U_1 or U_2, with a higher representation power for that entity.

As we experimentally verify in Section 5, this selection strategy prevents the row with the lesser energy to act as *noise* on the description of the corresponding entity and, thus, leads to significantly higher decomposition accuracies. Moreover, as the experiments show, the accuracy gains get higher as we target higher ranking decompositions that maintain more details by seeking a larger number of patterns in the data.

ALGORITHM 6: The outline of the Distributed Multi-task TensorDecomposition, D − M2TD, process

```
Input: Tensor $\mathcal{X}_1$, $\mathcal{X}_2$, Rank for each mode $r_1, r_2, \ldots, r_N$
Output: Factor Matrices $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}$ and core tensor $\mathcal{G}$
for the join tensor $\mathcal{J}$

    (1) Phase 1: Parallel decomposition of $\mathcal{X}_1$ and $\mathcal{X}_2$ to generate $U^{1(n)}, U^{2(n)}, n \in \{1, \ldots, N\}$
    (2) Phase 2: Parallel JE-Stitching $\mathcal{X}_1$, $\mathcal{X}_2$ to obtain the decomposition of the joined tensor $\mathcal{J}$
    (3) Phase 3: for $1 \leq n \leq N$
        (a) Parallel tensor matrix mutiplication- $\mathcal{G}_n = \mathcal{J} \times_n \mathbf{U}^{(n)}$
    (4) Return Factor Matrices $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}$ and core $\mathcal{G}$
```

4.5.4 Distributed M2TD (D-M2TD). A major challenge with tensor decomposition is its computational and space complexity. This is especially true for the Tucker decomposition with a dense core. In this section, relying on several key properties of the M2TD algorithm, we propose a three-phase distributed version of M2TD that can be efficiently and scalably executed on MapReduce or Spark-based platforms (see Algorithm 6):

• **Phase 1: Parallel Sub-tensor Decomposition:** Consider the M2TD-SELECT pseudocode in Algorithm 4. Here, $\mathcal{X}_1$ and $\mathcal{X}_2$ are two sub-tensors corresponding to two sub-systems constructed through PF-partitioning. These low-order sub-tensors can be decomposed (in parallel) independently from each other. Therefore, this phase can be parallelized using, for example, the popular distributed computing framework, MapReduce, using the following `map` and `reduce` operators:

- **map:** $\langle \boldsymbol{\kappa}, \rho_1, \rho_2, \ldots, \rho_M, \mathcal{X}_{\boldsymbol{\kappa}}(\rho_1, \rho_2, \ldots, \rho_M)\rangle$ on $\boldsymbol{\kappa}$. Here, $\boldsymbol{\kappa}$ is the low-order tensor ID; i.e., $\boldsymbol{\kappa} \in \{1, 2\}$. $\rho_1, \rho_2, \ldots, \rho_M$ together give the coordinate of a cell in the low-order tensor $\mathcal{X}_{\boldsymbol{\kappa}}$. Key-value pairs with the same κ are shuffled to the same reducer in the form of $\langle key : \boldsymbol{\kappa}, val : \rho_1, \rho_2, \ldots, \rho_M, \mathcal{X}_{\boldsymbol{\kappa}}(\rho_1, \rho_2, \ldots, \rho_M)\rangle$.
- **reduce:** $\langle key{:}\boldsymbol{\kappa}, val{:}\rho_1, \rho_2, \ldots, \rho_M, \mathcal{X}_{\boldsymbol{\kappa}}(\rho_1, \rho_2, \ldots, \rho_M)\rangle$. The reducer processing the key, $\boldsymbol{\kappa}$, receives the non-zero elements of sub-tensor $\mathcal{X}_{\boldsymbol{\kappa}}$ and decomposes it into sub-factor $\boldsymbol{U}^{\boldsymbol{\kappa}(\boldsymbol{n})}$, where $\boldsymbol{n}$ is the mode ID, by using SVD. Finally, reducer appropriately relabels each $\boldsymbol{U}^{\boldsymbol{\kappa}(\boldsymbol{n})}$ as $\boldsymbol{U}^{(\boldsymbol{n})}$ and emits each sub-factor as an independent file, with content $\langle key : n, value : i, j, \boldsymbol{U}^{(\boldsymbol{n})}(i, j)\rangle$. Here, i, j are the coordinates of sub-factor $\boldsymbol{U}^{(\boldsymbol{n})}$.

Note that this step can be further parallelized by leveraging parallel Tucker decomposition techniques, such as References [20, 36].

• **Phase 2: Parallel JE-stitching to Obtain Join Tensor, $\mathcal{J}$:** The goal of the stitching process is to increase the effective density of the ensemble. JE-stitching achieves this by joining the two sub-systems along their shared pivot modes to obtain the $\mathcal{J}$ tensor:

- **map:** $\langle \boldsymbol{\kappa}, \rho_1, \rho_2, \ldots, \rho_M, \mathcal{X}_{\boldsymbol{\kappa}}(\rho_1, \rho_2, \ldots, \rho_M)\rangle$. Key-value pairs with the same pivot mode index $(\rho_1, \rho_2, \ldots, \rho_k)$ are shuffled to the same reducer in the form of $\langle key{:}(\rho_1, \rho_2, \ldots, \rho_k)$, $val{:}\rho_1, \rho_2, \ldots, \rho_M, \mathcal{X}_{\boldsymbol{\kappa}}(\rho_1, \rho_2, \ldots, \rho_M)\rangle$.

- **reduce:** $\langle key{:}(\rho_1, \rho_2, \ldots, \rho_k),\ val{:}\rho_1, \rho_2, \ldots, \rho_M, \boldsymbol{\mathcal{X}_\kappa}(\rho_1, \rho_2, \ldots, \rho_M)\rangle$. The join ensemble $\mathcal{J}(\rho_1, \rho_2, \ldots, \rho_k, \ldots)$ is constructed for all pairs of $\boldsymbol{\mathcal{X}_\kappa}$, that agree on the parameter values for the k pivot parameters.

• **Phase 3: Parallel Tensor-matrix Multiplication to Recover the Core:** As we see in Section 5, the costliest part of the algorithm is the final step where the join tensor $\mathcal{J}$ is multiplied by the transposes of the factor matrices to recover the dense core. We parallelize this as follows:

- **map:** $\langle \rho_1, \rho_2, \ldots, \rho_N, \mathcal{J}(\rho_1, \rho_2, \ldots, \rho_N)\rangle$, $\langle \boldsymbol{n}, i, j, \boldsymbol{U}^{(n)}(i,j)\rangle$. Cells of $\mathcal{J}$ (from Phase 2) with index $(\rho_1, \rho_2, \ldots, \rho_{n-1}, \rho_{n+1}, \ldots, \rho_N)$ are shuffled to the same reducer in the form of $\langle key{:}(\rho_1, \rho_2, \ldots, \rho_{n-1}, \rho_{n+1}, \ldots, \rho_N), val{:}\mathcal{J}(\rho_1, \rho_2, \ldots, \rho_N)\rangle$
- **map:** $\langle \boldsymbol{n}, i, j, \boldsymbol{U}^{(n)}(i,j)\rangle$. Outputs of Phase 1 $\langle \boldsymbol{n}, i, j, \boldsymbol{U}^{(n)}(i,j)\rangle$ are shuffled to the same reducer based on mode ID n in the form of $\langle key{:}(\rho_1, \rho_2, \ldots, \rho_{n-1}, \rho_{n+1}, \ldots, \rho_N), val{:}\boldsymbol{n}, i, j, \boldsymbol{U}^{(n)}(i,j)\rangle$
- **reduce:** The reducer takes

$$\langle key : (\rho_1, .., \rho_{n-1}, \rho_{n+1}, .., \rho_N), val : \mathcal{J}(\rho_1, , \ldots, \rho_N)\rangle$$

and

$$\langle key : (\rho_1, .., \rho_{n-1}, \rho_{n+1}, .., \rho_N), val : \boldsymbol{n}, i, j, \boldsymbol{U}^{(n)}(i,j)\rangle$$

and performs vector-matrix multiplication to emit $\langle(\rho_1, \rho_2, \ldots, \rho_{n-1}, j, \rho_{n+1}, \ldots, \rho_N), \sum_{\rho_n=1}^{I_n} \mathcal{J}(\rho_1, , \ldots, \rho_n, \ldots, \rho_N) * \boldsymbol{U}^{(n)}(\rho_n, j)\rangle$.

In the next section, we investigate the impact of this parallelization approach to the performance of the proposed partition-stitch sampling through M2TD decomposition.

5 EXPERIMENTS

In this section, we report results of the experiments that aim to assess the effectiveness and efficiency of the proposed partition-stitch ensemble sampling strategy and the novel M2TD scheme. For these experiments, we used the Chameleon cloud platform [22]: We deployed all algorithms on 18 xxlarge instances, with 8-core vCPU, 32 GB memory, 160 GB disk space. Distributed versions were implemented in Java 8, over Hadoop 2.7.3. The key system parameters and their value ranges are reported in Table 1 and explained below.

5.1 Dynamic Systems

In these experiments, we consider three dynamic processes: *double pendulum*, triple pendulum, lorenz system [9]. The code for these systems was obtained from References [13] and [9]. These dynamic processes are selected for their varying complexities:

The **double pendulum** system has four parameters: initial angle, ϕ_1, and weight, $m1$, of the first pendulum as well as the initial angle, ϕ_2, and weight, $m2$, of the second pendulum.

The **triple pendulum (with variable friction)** system is similar, but more complex due to the addition of a third pendulum. Moreover, the system has a different set of initial parameters: the angle ϕ_1 of the first pendulum, the initial angle ϕ_2 of the second pendulum, the initial angle ϕ_3 of the third pendulum, and the friction f of whole system. Intuitively, unlike the double pendulum system, in the triple pendulum system the friction is considered as a simulation parameter.

The **Lorenz system** is notable for having chaotic solutions for certain initial conditions [9]. The system has four variable parameters: the coordinate of the initial position, z, and three other system parameters, σ, β, ρ.

Table 1. Experiment Setup—Default Values, Used Unless Otherwise Specified, Are Highlighted

	Alternative values		
Dynamic systems	**Double Pend.**;	Triple Pend.	Lorenz System
Parameter resolution	60 ;	**70**;	80
Size of the corresponding simulation space (S)	$60^5 (8 \times 10^8)$;	$\mathbf{70^5 (2 \times 10^9)}$;	$80^5 (3 \times 10^9)$
Pivot density (P)	10%;	**100%**	
Sub-system density (E)	10%;	**100%**	
Ensemble budget ($B = 2 \times P \times E \times S$)	4×10^4, 4×10^5,	7×10^4, 7×10^5,	1×10^5, 1×10^6
Target decomposition rank (r)	5;	**10**;	20
Stitching technique	**Join**;	Zero-Join	
Number of servers	2, 6, 10, 14, **18**		

Finally, we use an **epidemic simulation** dataset created using STEM [12]: We consider a scenario, where an epidemic is occurring in the U.S. We use SEIR model with three input parameters (transmission, recovery, and mortality rates), each parameter is a continuous real number ranging from 0 to 1, and we consider 40 distinct values for each parameter, i.e., they all vary from 0.025 to 1 with increment of 0.025. As the outcome, we focus on the number of deaths over time.

5.2 Simulation Ensembles

For the above systems, we construct five-mode simulation ensembles. Each cell of the five-mode ensemble simulation tensor encodes the Euclidean distance between the states of the resulting simulated system and the observed system parameters at a given timestamp for a given quadruple of simulation parameters. Intuitively, each cell encodes the relationship between a given simulation instance to a configuration observed in the real world.

As we see in Table 1, in the experiments, the size of the simulation space varied between $60^5 \sim 8 \times 10^8$ to $80^5 \sim 3 \times 10^9$ simulation instances. In contrast, the simulation instance budgets were on the order of 10^4 to 10^5, indicating that, despite the large number of simulations included in the ensembles, the resulting ensemble tensors were very sparse (densities on the order of $\sim 10^{-4}$). Despite this sparsity, for the different configurations considered in Table 1, the simulation ensemble required from 25 GB to 105 GB data storage.

5.3 Alternative Ensemble Construction Schemes

In this section, we evaluated the M2TD-AVG, -CONCAT, and -SELECT strategies and compared them against the conventional (Random, Grid, and Slice) ensemble sampling approaches (Section 4.2). For M2TD-based schemes, we considered the case with a single pivot parameter and, to analyze worst-case behavior, we sampled the sub-systems randomly.

5.4 Evaluation Criteria

We compared accuracy and efficiency of alternative schemes, for different target decomposition ranks, different parameter space resolutions, and simulation budgets (see Table 1). To measure accuracy, we use the Frobenius norm of the difference tensor (see Section 4.1). We also report the decomposition times.

Table 2. Results for Different Resolutions

Resolution	Rank	(a) Accuracy						(b) Decomposition Time (seconds)					
		M2TD			Conventional			M2TD			Conventional		
		AVG	CONCAT	SELECT	Random	Grid	Slice	AVG	CONCAT	SELECT	Random	Grid	Slice
60	5	0.49	0.49	**0.54**	1E-8	3E-4	2E-4	808	797	785	203	144	167
	10	0.50	0.50	**0.62**	2E-7	3E-4	2E-4	808	819	849	234	148	186
	20	0.52	0.53	**0.56**	5E-6	3E-4	2E-4	1,034	929	935	348	456	258
70	5	0.46	0.46	**0.51**	7E-9	2E-4	2E-4	1,508	1,581	1,594	315	209	193
	10	0.47	0.48	**0.57**	9E-8	2E-4	2E-4	1,696	1,645	1,576	379	201	244
	20	0.49	0.50	**0.73**	2E-6	2E-4	2E-4	1,866	1,914	1,995	575	744	381
80	5	0.46	0.46	**0.50**	4E-9	1E-4	1E-4	3,990	3,591	4,907	414	227	336
	10	0.47	0.47	**0.49**	4E-8	1E-4	1E-4	5,232	5,979	6,068	514	239	410
	20	0.48	0.49	**0.59**	1E-6	2E-4	1E-4	5,341	5,707	5,439	860	883	606

Double Pendulum, Pivot = t, P = 100%, E = 100%.

5.5 Discussions of the Results

5.5.1 General Overview. Table 2 focuses on the double pendulum system and compares accuracies and decomposition times for various approaches considered in this article for the different target ranks and for different parameter resolutions. As we see in the table, the M2TD-based algorithms provide several orders better accuracy than the conventional approaches, with the same number of simulation instances. As expected, among the conventional schemes, the Random strategy provides the worst and the Grid strategy provides the best accuracy; however, even Grid is $\sim$1,000$\times$ worse than the proposed M2TD-SELECT algorithm. As also expected, among the M2TD-based algorithms, M2TD-SELECT provides the best overall accuracy: moreover, the relative performance gains of M2TD-SELECT algorithm further increases for larger decomposition ranks, indicating that as we seek more detailed patterns in the ensemble, M2TD-SELECT better captures these underlying patterns in the data.

In the table, we also see that M2TD-based algorithms are somewhat more expensive than the conventional sampling strategies; but the gains in accuracy are several orders higher than the decomposition time overheads of M2TD-based techniques. This is because, as highlighted in Section 4.4.2, the proposed partition-stitch technique increases the *effective density* of the join ensemble. Consequently, the increase in the decomposition is well amortized by the increase in the effective simulation density. In these experiments, each double pendulum simulation took roughly 0.66ms. Given this, obtaining an ensemble simulation with density $70^4 (= 70^2 \times 70^2)$ would require roughly 16,000 seconds (ignoring the additional time to decompose). In contrast, the proposed M2TD-based techniques are able to achieve the same *effective density* by running only 2×70^2 simulations in just 46 seconds and obtain the ensemble decomposition in an additional $\sim$1,600 seconds. This points to the impressive performance gains provided by the proposed M2TD technique.

One question that remains is whether we could have joined the sub-ensembles directly into tensor $\mathcal{J}$ to decompose instead of relying on the M2TD techniques: the answer to this question is a *strong no*: For the experiments reported in Table 2, with the configuration of 18 xxlarge servers, direct decomposition of the resulting dense tensor was not feasible due to memory limitations.

5.5.2 Decomposition Time Distribution. Table 3 presents how the decomposition time is split among the three phases of the map-reduce process described in Section 4.5.4. The table also shows how the execution time varies as we change the number of servers allocated for the decomposition

Table 3. Results by Server Count

Number of Servers	Decomposition Time (seconds)					
	M2TD-SELECT			Conventional		
	Phase 1	Phase 2	Phase 3	Random	Grid	Slice
2	52	817	4,167	670	420	488
6	62	383	1,802	464	275	318
10	61	371	1,318	415	237	280
14	65	354	1,279	381	214	253
18	67	363	1,118	379	201	244

Double Pendulum, Resolution = 70, Rank = 10, Pivot = t, P = 100%, E = 100%.

Table 4. Results for Different Dynamical Systems

Dynamical System	(a) Accuracy						(b) Decomposition Time (seconds)					
	M2TD			Conventional			M2TD			Conventional		
	AVG	CONCAT	SELECT	Random	Grid	Slice	AVG	CONCAT	SELECT	Random	Grid	Slice
Double P.	0.47	0.48	**0.57**	9E-8	2E-4	2E-4	1,696	1,645	1,576	379	201	244
Triple P.	0.25	0.25	**0.31**	6E-8	2E-4	1E-4	992	1,422	1,106	221	180	166
Lorenz S.	0.31	0.32	**0.36**	4E-8	2E-4	1E-4	1,728	1,850	1,705	444	230	211
Epidemic S.	**0.30**	0.28	0.26	0.026	0.027	0.026	0.431	0.421	0.421	0.018	0.024	0.018

Resolution = 70, Rank = 10, Pivot = t, P = 100%, E = 100%.

process. As we see in this table, as expected, the third phase where we recover the core tensor of the decomposition is the costliest step of the process. We also see that allocating more servers indeed helps bring the cost of this phase down; however, there are diminishing returns due to data communication overheads.

5.5.3 Varying Data Sets. In Table 4, we study the accuracy and decomposition time results for different dynamic systems. As we see here, also for the triple pendulum and Lorenz systems, we observe the very same pattern: M2TD-SELECT provides the best accuracy among all alternatives, providing several orders of magnitude gain in accuracy relative to the conventional schemes. M2TD is once again one order superior in accuracy than the conventional techniques for the epidemic dataset, while providing acceptably low decomposition times: note that since this dataset has three modes, the decomposition times are significantly faster than the decomposition times needed by the five-mode pendulum and Lorenz systems. It is also interesting that, in this case, M2TD-AVG is able to provide the best accuracy among all M2TD variants.

5.5.4 Varying Budgets and Zero-Joins. In the default experiments considered above, the budget was selected such that the sub-ensembles would have a perfect density of 1.0. In the first row of Table 5, we reduced the ensemble budget by taking 1/10th of the samples we considered in the previous examples. Naturally, this results in a drop in accuracy for all approaches. However, M2TD-based schemes remain several orders better than the conventional approaches.

The table also shows that when the budgets (thus, sub-ensemble densities) are low, we can boost the overall accuracy by leveraging zero-joins (introduced in Section 4.4.2), rather than using simple joins when implementing JE-stitching.

5.5.5 Varying Pivot/Sub-ensemble Densities. Tables 6 and 7 show the impact of reduced pivot and sub-ensemble densities (i.e., P and E), respectively. As we see here, the overall pattern is as

Table 5. Results for Different Ensemble Budgets

Budget	(a) Accuracy						(b) Decomposition Time (seconds)					
	M2TD			Conventional			M2TD			Conventional		
	AVG	CONCAT	SELECT	Random	Grid	Slice	AVG	CONCAT	SELECT	Random	Grid	Slice
4×10^4 (join)	3.5E-5	3.4E-5	**4.1E-5**	9E-9	2E-5	2E-6	200	201	200	190	175	183
4×10^4 (zero-join)	3.3E-3	3.2E-3	**3.9E-3**	9E-9	2E-5	2E-6	596	598	592	190	175	183
4×10^5	0.47	0.48	**0.57**	9E-8	2E-4	2E-4	1,696	1,645	1,576	379	201	244

Double pendulum, resolution = 70, rank = 10, pivot = t; note that $B = 4 \times 10^5$ corresponds to the case where both pivot, P, and sub-systems, E, have 100% densities.

Table 6. Results for Different Pivot Densities

P Density	(a) Accuracy						(b) Decomposition Time (seconds)					
	M2TD			Conventional			M2TD			Conventional		
	AVG	CONCAT	SELECT	Random	Grid	Slice	AVG	CONCAT	SELECT	Random	Grid	Slice
10%	3.5E-2	7.6E-3	**3.6E-2**	9E-9	2E-5	2E-6	606	897	607	190	175	183
100%	0.47	0.48	**0.57**	9E-8	2E-4	2E-4	1,696	1,645	1,576	379	201	244

Double Pendulum, Resolution = 70, Rank = 10, Pivot = t, E = 100%.

Table 7. Results for Different Densities

E Density	(a) Accuracy						(b) Decomposition Time (seconds)					
	M2TD			Conventional			M2TD			Conventional		
	AVG	CONCAT	SELECT	Random	Grid	Slice	AVG	CONCAT	SELECT	Random	Grid	Slice
10% (join)	4E-5	4E-5	**4.5E-5**	9E-9	2E-5	2E-6	207	202	201	190	175	183
10% (zero-join)	3.4E-3	3.3E-3	**3.8E-3**	9E-9	2E-5	2E-6	602	640	617	190	175	183
100%	0.47	0.48	**0.57**	9E-8	2E-4	2E-4	1,696	1,645	1,576	379	201	244

Double Pendulum, Resolution = 70, Rank = 10, Pivot = t, P = 100%.

before: Reduction in the simulation budget reduces the overall accuracy; however, M2TD-based schemes provide significantly higher accuracy overall.

An interesting observation, however, is that (while the total number of simulations is the same) reduction in the pivot sub-ensemble density has a significantly higher impact than the reduction in the pivot density. This is because, as discussed in Section 4.4.2, the effective density of a stitched simulation ensemble is proportional to $P \times E^2$, and thus reductions in E have a more significant impact than reductions in P. This further confirms our initial hypothesis that maintaining sub-ensemble densities high is important for accurate characterization of the system being studied.

5.5.6 Selection of the Pivot Parameter. In Table 8, we vary the pivot parameter[2]: As we expected, which parameter is selected as the pivot has some impact on the accuracy of the proposed partition-stitch scheme. However, whichever pivot is selected, the overall accuracy is several orders of magnitude better than that of conventional schemes, indicating that we do not need very precise information about the system being studied to decide how to partition the system.

5.6 Sub-actor Execution Times

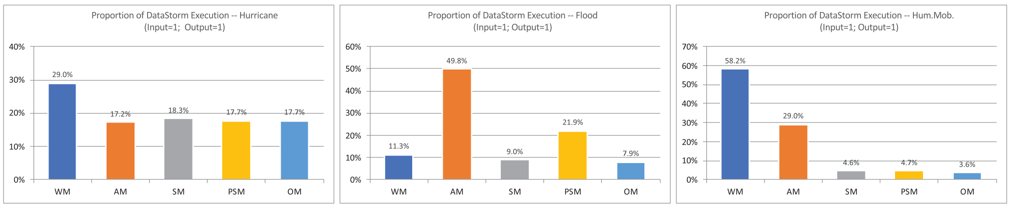
As we have discussed in Section 3, each `DataStorm` actor consists of multiple sub-actors. In this section, we consider a sample `DataStorm` workflow with the model dependency graph shown

[2]Due to space constraints, we omit experiments where we keep the same pivot parameter, but vary the groupings of free parameters. The results are similar to the results of pivot parameter selection.

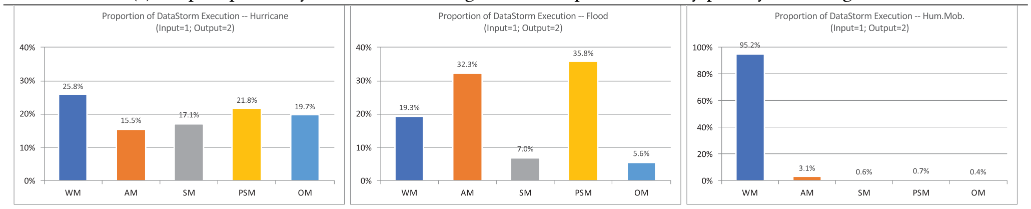
Table 8. Results for Different Pivots

Pivot	(a) Accuracy						(b) Decomposition Time (seconds)					
	M2TD			Conventional			M2TD			Conventional		
	AVG	CONCAT	SELECT	Random	Grid	Slice	AVG	CONCAT	SELECT	Random	Grid	Slice
t	0.47	0.48	**0.57**	9E-8	2E-4	2E-4	1,696	1,645	1,576	379	201	244
ϕ_1	0.35	0.36	**0.40**				1,607	1,673	1,673			
ϕ_2	0.40	0.41	**0.56**				1,694	1,677	1,571			
m_1	0.58	0.59	**0.71**				1,661	1,512	1,697			
m_2	0.41	0.40	**0.42**				1,556	1,602	1,538			

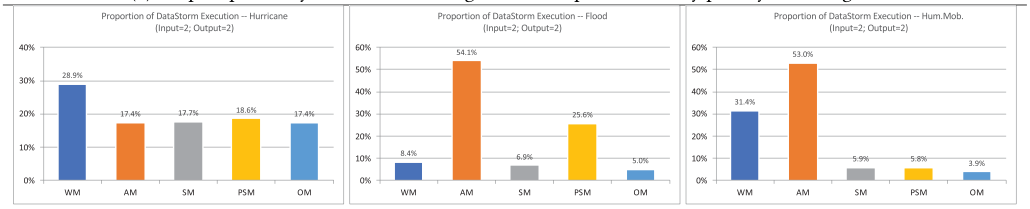
Double pendulum, resolution = 70, rank = 10, $P = 100\%$, $E = 100\%$; three-mode sub-systems are created such that free parameters of the same pendulum are kept in the same sub-system.



(a) # inputs picked by the window manager = 1; # outputs created by post-sync. manager = 1



(b) # inputs picked by the window manager = 1; # outputs created by post-sync. manager = 2



(c) # inputs picked by the window manager = 2; # outputs created by post-sync. manager = 2

Fig. 16. Sub-actor execution time percentages for different models and different input output scenarios—WM: window manager; AM: alignment manager; SM: sampling manager; PSM: post-synchronization manager; OM: output manager; note that the time taken by the execution manager (which distributed the tasks on the available servers and runs the actual simulations) is excluded.

in Figure 4 and we investigate the amount (percentage) of time spent on various sub-actors—excluding the execution manager whose execution depends on the actual model that is being simulated and the specific simulator that is being used. This three-model DataStorm workflow was executed on a distributed configuration with 15 servers allocated for model execution. In the charts included in Figure 16, we vary the number of windowing sets (Definition 13, here referred to as inputs) picked by the window manager for sampling and the number of outputs created by the post-synchronization manager.

As we see in the figure, the window manager, alignment manager, and post-synchronization manager take the lion's share of the work done by DataStorm actors.

For the source model (Hurricane) of the workflow, the work is uniformly distributed, except for the window manager, which needs time to prepare data required by the first model to execute.

For the second model (Flood) in the workflow, the alignment manager takes substantial time to align and rescale the data generated by the hurricane simulator to bring it into a format usable by the flood model. The work of the alignment manager increases proportionate to the number of windowing sets (inputs) it receives.

For the final model (Human Mobility) in the workflow, the window manager takes substantially longer, as it consumes data from two distinct upstream models—as selecting most compatible data for constructing temporal windows is costly, when both previous models return more than one output each, more computation is required to find optimal windowing sets. As expected, the work of the alignment manager is highest when the window manager picks two windowing sets to be processed, increasing amount of computation necessary in the alignment phase.

6 CONCLUSIONS

Data- and model-driven computer simulations are increasingly critical in urban systems, including emergency planning and response. Simulation-based decision-making, however, introduces several fundamental data challenges [31, 39]: Many complex processes (such as disasters [8]) involve various distinct, yet inter-dependent, sub-processes. Consequently, to be useful, these simulations may track hundreds of parameters, spanning multiple layers and spatial-temporal frames, affected by complex inter-dependent dynamic processes. Moreover, due to large numbers of unknowns, decision-makers usually need to generate an ensemble of stochastic realizations, requiring thousands of individual simulation instances, each with different parameter settings corresponding to different, but plausible, scenarios.

In this article, we presented a novel `DataStorm` system that aims to support integration of existing simulation, analysis, data processing, and visualization components into integrated urban simulation/analysis workflows. `DataStorm` provides a spectrum of data services, including (a) model/data integration and alignment, (b) distributed/parallel workflow orchestration, (c) parameter space sampling and simulation ensemble creation, (d) ensemble storage, search, analysis, visualization, and sharing. Recognizing that simulation ensembles are inherently sparse and this sparsity leads to poor decision-making, we also presented a density-boosting partition-stitch sampling scheme to help increase the effective density of the simulation samples to boost accuracy. We have complemented this sampling scheme with Multi-Task Tensor Decomposition (M2TD) to efficiently stitch these sub-ensembles in a way that leverages partial and imperfect knowledge from partial dynamical systems to effectively obtain a global view of the complex process being simulated. Experiment results showed the efficiency and the effectiveness of this approach relative to more conventional techniques for constructing simulation ensembles.

REFERENCES

[1] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludäscher, and Stephen Mock. 2004. Kepler: An extensible system for design and execution of scientific workflows. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM'04)*, Vol. 16. 423–424. DOI: https://doi.org/10.1109/SSDBM.2004.44

[2] Ilkay Altintas, Jianwu Wang, Daniel Crawl, and Weizhong Li. 2012. Challenges and approaches for distributed workflow-driven analysis of large-scale biological data: Vision paper. In *Proceedings of the Joint EDBT/ICDT Workshops (EDBT-ICDT'12)*. ACM, New York, NY, 73–78. DOI: https://doi.org/10.1145/2320765.2320791

[3] Duygu Balcan, Bruno Gonçalves, Hao Hu, José J. Ramasco, Vittoria Colizza, and Alessandro Vespignani. 2010. Modeling the spatial spread of infectious diseases: The GLobal epidemic and mobility computational model. *J. Comput. Sci.* 1, 3 (Aug. 2010), 132–145. DOI: https://doi.org/10.1016/j.jocs.2010.07.002

[4] Michael Batty. 2013. Visually-driven urban simulation: Exploring fast and slow change in residential location. *Environ. Plan. A: Econ. Space* 45, 3 (2013), 532–552.

[5] Hans Walter Behrens, K. Selçuk Candan, Xilun Chen, Ashish Gadkari, Yash Garg, Mao-Lin Li, Xinsheng Li, Sicong Liu, Nicholas Martinez, Jiayong Mo, Elliot Nester, Silvestro Poccia, Manjusha Ravindranath, and Maria Luisa Sapino. 2018. Datastorm-FE: A data- and decision-flow and coordination engine for coupled simulation ensembles. *Proc. VLDB Endow.* 11, 12 (Aug. 2018), 1906–1909.

[6] Hans Walter Behrens, Mao-Lin Li, Ashish Gadkari, Yash Garg, Xilun Chen, Sicong Liu, and K. Selçuk Candan. 2019. Load-adaptive continuous coupled-simulation ensembles with DataStorm and Chameleon. In *Proceedings of the Chameleon User Meeting*.

[7] Chun-Hung Chen and Loo Hay Lee. 2010. *Stochastic Simulation Optimization - An Optimal Computing Budget Allocation*. System Engineering and Operations Research, Vol. 1. WorldScientific.

[8] Committee on Environment and Natural Resources. 2008. Grand challenges for disaster reduction. *Executive Office of the President* (2008). https://www.sdr.gov/docs/SDRGrandChallengesforDisasterReduction.pdf.

[9] Greg Cope. 2017. Programming the Lorenz Attractor. Retrieved from https://www.algosome.com/articles/lorenz-attractor-programming-code.html

[10] Laurent Guillaume Courty, Adrián Pedrozo-Acuña, and Paul David Bates. 2017. Itzï (Version 17.1): An open-source, distributed GIS model for dynamic flood simulation. *Geosci. Model Dev.* 10, 4 (May 2017), 1835–1847. DOI: https://doi.org/10.5194/gmd-10-1835-2017

[11] Christopher Davis, Wei Wang, Shuyi S. Chen, Yongsheng Chen, Kristen Corbosiero, Mark DeMaria, Jimy Dudhia, Greg Holland, Joe Klemp, John Michalakes, Heather Reeves, Richard Rotunno, Chris Snyder, and Qingnong Xiao. 2008. Prediction of landfalling hurricanes with the advanced hurricane WRF model. *Month. Weath. Rev.* 136, 6 (June 2008), 1990–2005. DOI: https://doi.org/10.1175/2007MWR2085.1

[12] Stefan B. Edlund, Matthew A. Davis, and James H. Kaufman. 2010. The spatiotemporal epidemiological modeler. In *Proceedings of the 1st ACM International Health Informatics Symposium (IHI'10)*. ACM, New York, NY, 817–820. DOI: https://doi.org/10.1145/1882992.1883115

[13] Christoph Federrath. 2017. Java Applets. Retrieved from http://www.ita.uni-heidelberg.de/~chfeder/applets.shtml?lang=en.

[14] Ronald Aylmer Fisher. 1935. *The Design of Eexperiments*. Oliver and Boyd.

[15] Ignacio Garcia-Dorado, Daniel G. Aliaga, Saiprasanth Bhalachandran, Paul Schmid, and Dev Niyogi. 2017. Fast weather simulation for inverse procedural design of 3D urban models. *ACM Trans. Graph.* 36, 2 (2017), 21:1–21:19.

[16] Michael Gollner, Arnaud Trouve, Ilkay Altintas, Jessica Block, Raymond de Callafon, Craig Clements, Anna Cortes, Evan Ellicott, Jean Baptiste Filippi, Mark Finney, Kayo Ide, Mary Ann Jenkins, Daniel Jimenez, Christopher Lautenberger, Jan Mandel, Melanie Rochoux, and Albert Simeoni. 2015. Towards data-driven operational wildfire spread modeling: A report of the NSF-funded WIFIRE workshop. (Jan. 2015). DOI: https://doi.org/10.13016/M2Z70R

[17] Richard A. Harshman. 1970. Foundations of The PARAFAC procedure: Models and conditions for an 'explanatory" multimodal factor analysis. In *UCLA Working Papers in Phonetics*.

[18] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole Goble, Mathew R. Pocock, Peter Li, and Tom Oinn. 2006. Taverna: A tool for building and running workflows of services. *Nucleic Acids Res.* 34, Web Server issue (July 2006), W729–W732. DOI: https://doi.org/10.1093/nar/gkl320

[19] C. Jelesnianski, Jye Chen, W. Shaffer, and A. Gilad. 1984. SLOSH—A hurricane storm surge forecast model. In *Proceedings of the OCEANS 1984 Conference*. 314–317. DOI: https://doi.org/10.1109/OCEANS.1984.1152341

[20] I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos. 2015. HaTen2: Billion-scale tensor decompositions. In *Proceedings of the IEEE 31st International Conference on Data Engineering*. 1047–1058. DOI: https://doi.org/10.1109/ICDE.2015.7113355

[21] N. L. Johnson. 1961. Sequential analysis: A survey. *J. Roy. Statist. Soc. Series A (General)* 124, 3 (1961), 372–411. DOI: 10.2307/2343243

[22] Kate Keahey, Pierre Riteau, Dan Stanzione, Tim Cockerill, Joe Mambretti, Paul Rad, and Paul Ruth. 2018. Chameleon: A scalable production testbed for computer science research. In *Contemporary High Performance Computing: From Petascale toward Exascale*. CRC Press.

[23] Katarzyna Keahey, Mauricio Tsugawa, Andrea Matsunaga, and Jose Fortes. 2009. Sky computing. *IEEE Internet Comput.* 13, 5 (2009), 43–51.

[24] Ari Keranen. 2008. Opportunistic network environment simulator. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques (SIMUTools'09)*. https://www.netlab.tkk.fi/tutkimus/dtn/theone/pub/the_one.pdf.

[25] Mijung Kim and K. Selçuk Candan. 2016. Decomposition-by-normalization (DBN): Leveraging approximate functional dependencies for efficient CP and Tucker decompositions. *Data Mining Knowl. Discov.* 30, 1 (Jan. 2016), 1–46. DOI: https://doi.org/10.1007/s10618-015-0401-6.

[26] Stefan Klus, Patrick Gelß, Sebastian Peitz, and Christof Schütte. 2018. Tensor-based dynamic mode decomposition. *Nonlinearity* 31, 7 (June 2018), 3359–3380. DOI: https://doi.org/10.1088/1361-6544/aabc8f.

[27] T. Kolda and B. Bader. 2009. Tensor decompositions and applications. *SIAM Rev.* 51, 3 (Aug. 2009), 455–500. DOI: https://doi.org/10.1137/07070111X.

[28] T. G. Kolda and J. Sun. 2008. Scalable tensor decompositions for multi-aspect data mining. In *Proceedings of the 8th IEEE International Conference on Data Mining*. 363–372. DOI: https://doi.org/10.1109/ICDM.2008.89.

[29] X. Li, K. S. Candan, and M. L. Sapino. 2018. M2TD: Multi-task tensor decomposition for sparse ensemble simulations. In *Proceedings of the IEEE 34th International Conference on Data Engineering (ICDE'18)*. 1144–1155.

[30] X. Li, S. Huang, K. S. Candan, and M. L. Sapino. 2016. 2PCP: Two-phase CP decomposition for billion-scale dense tensors. In *Proceedings of the IEEE 32nd International Conference on Data Engineering (ICDE'16)*. 835–846.

[31] Sicong Liu, Yash Garg, K. Selcuk Candan, Maria Luisa Sapino, and Gerardo Chowell-Puente. 2015. NOTES2: Networks-of-traces for epidemic spread simulations. In *Proceedings of the Workshops at the 29th AAAI Conference on Artificial Intelligence*.

[32] Sicong Liu, Silvestro Poccia, K. Selçuk Candan, Gerardo Chowell, and Maria Luisa Sapino. 2016. epiDMS: Data management and analytics for decision-making from epidemic spread simulation ensembles. *J. Infect. Dis.* 214, suppl_4 (2016), S427–S432.

[33] C. D. Tharindu Mathew, Paulo R. Knob, Soraia Raupp Musse, and Daniel G. Aliaga. 2019. Urban walkability design using virtual population simulation. *Comput. Graph. Forum* 38, 1 (2019), 455–469.

[34] Panayiotis Neophytou, Panos K. Chrysanthis, and Alexandros Labrinidis. 2011. CONFLuEnCE: CONtinuous workFLow ExeCution Engine. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'11)*. ACM, New York, NY, 1311–1314. DOI: https://doi.org/10.1145/1989323.1989485

[35] K. B. Olsen, S. M. Day, L. A. Dalguer, J. Mayhew, Y. Cui, J. Zhu, V. M. Cruz-Atienza, D. Roten, P. Maechling, T. H. Jordan, D. Okaya, and A. Chourasia. 2009. ShakeOut-D: Ground motion estimates using an ensemble of large earthquakes on the Southern San Andreas fault with spontaneous rupture propagation. *Geophys. Res. Lett.* 36, 4 (2009). DOI: https://doi.org/10.1029/2008GL036832

[36] Evangelos E. Papalexakis, Christos Faloutsos, and Nicholas D. Sidiropoulos. 2012. ParCube: Sparse parallelizable tensor decompositions. In *Machine Learning and Knowledge Discovery in Databases (Lecture Notes in Computer Science)*, Peter A. Flach, Tijl De Bie, and Nello Cristianini (Eds.). Springer Berlin, 521–536.

[37] Giulia Pedrielli, Yinchao Zhu, and Loo Hay Lee. 2015. Single-run simulation optimization through time dilation and optimal computing budget allocation. http://www.smmso.org/SMMSO2015/Downloads/Giulia%20Pedrielli.pdf.

[38] Giulia Pedrielli, Yinchao Zhu, Loo Hay Lee, and Haobin Li. 2016. Empirical analysis of the performance of variance estimators in sequential single-run ranking & selection: The case of time dilation algorithm. In *Proceedings of the Winter Simulation Conference (WSC'16)*. IEEE Press, Piscataway, NJ, 738–748.

[39] Silvestro Poccia, Maria Luisa Sapino, Sicong Liu, Xilun Chen, Yash Garg, Shengyu Huang, Jung Hyun Kim, Xinsheng Li, Parth Nagarkar, and Kasim Candan. 2017. SIMDMS: Data management and analysis to support decision making through large simulation ensembles. In *Proceedings of the 20th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'17)*. OpenProceedings.org, 582–585. DOI: https://doi.org/10.5441/002/edbt.2017.75.

[40] Mark Rogers, Lei Li, and Stuart J. Russell. 2013. Multilinear dynamical systems for tensor time series. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2634–2642.

[41] L. W. Schruben. 1997. Simulation optimization using simultaneous replications and event time dilation. In *Proceedings of the Winter Simulation Conference*. 177–180. DOI: https://doi.org/10.1109/WSC.1997.640394.

[42] EmitLab DataStorm Team. 2018. DataStorm demo, v1. Retrieved from https://www.youtube.com/watch?v=L74jwa78Dg4.

[43] EmitLab DataStorm Team. 2018. DataStorm release, v1. Retrieved from https://github.com/EmitLab/DataStorm-Release.

[44] Ledyard R. Tucker. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika* 31, 3 (Sept. 1966), 279–311. DOI: https://doi.org/10.1007/BF02289464.

[45] Paul Waddell. 2002. UrbanSim: Modeling urban development for land use, transportation, and environmental planning. *J. Amer. Plan. Assoc.* 68, 3 (2002), 297–314.

[46] Denis Weerasiri, Moshe Chai Barukh, Boualem Benatallah, Quan Z. Sheng, and Rajiv Ranjan. 2017. A taxonomy and survey of cloud resource orchestration techniques. *Comput. Surv.* 50, 2 (May 2017), 1–41.

[47] Ming Zhao, K. Selçuk Candan, Huan Liu, Hasan Davulcu, and Fengbo Ren. 2016. NSF Award Search: Award#1629888 - II-NEW: GEARS - An Infrastructure for Energy-Efficient Big Data Research on Heterogeneous and Dynamic Data. Retrieved from https://nsf.gov/awardsearch/showAward?AWD_ID=1629888

Received June 2019; revised November 2020; accepted January 2021