Active Workflow System for Near Real-Time Extreme-Scale Science

Yanwei Zhang[†] **Qing Liu**[‡] **Scott Klasky**[‡] **Matthew Wolf**[†,‡] **Karsten Schwan**[†]
Greg Eisenhauer[†] **Jong Choi**[‡] **Norbert Podhorszki**[‡]

[†]College of Computing, Georgia Institute of Technology, Atlanta, GA
[‡]Scientific Data Group, Oak Ridge National Laboratory, Oak Ridge, TN

yanwei.zhang@gatech.edu
{liuq, klasky}@ornl.gov
{mwolf, karsten.schwan, eisen}@cc.gatech.edu
{choij, pnorbert}@ornl.gov

Abstract

In recent years, streaming-based data processing has been gaining substantial traction for dealing with overwhelming data generated by real-time applications, from both enterprise sources and scientific computing. In this work, however, we look at an emerging class of scientific data with Near Real-Time (NRT) requirement, in which data is typically generated in a bursty fashion with the near real-time constraints being applied primarily between bursts, rather than within a stream. A key challenge for this types of data sources is that the processing time per data element is not uniform, and not always feasible to predict. Given the observations on the increasing unpredictability of compute load and system dynamics, this work looks to adapt streaming-based approach to the context of this new class of large experiments and simulations that have complex run-time control and analysis issues.

In particular, we deploy a novel two-tier scheme for handling the increasing unpredictability of runtime behaviors: Instead of relying on determining *what* and *where* to run the scientific workflows beforehand or partial dynamically, the decision will also be adaptively enhanced online according to system runtime status. This is enabled by embedding workflow along with data streams. Specifically, we break data outputs generated from experiments or simulations into multiple self-describing "chunks", which we call *active data objects*. As such, if there is a transient hotspot observed, a data object with unfinished workflow pipeline can break its previous schedule and search for a least loaded location to continue the execution. Our preliminary experiment results based on synthetic workloads demonstrate the proposed active workflow system as a very promising solution by outperforming the state-of-the-art semi-dynamic workflow schedulers with an improved workflow completion time, as well as a good scalability.

Categories and Subject Descriptors C.2.4 [*Distributed Systems*]: Distributed applications

Keywords Scientific workflow system; near real-time science; distributed workflow scheduler; stream processing; load balancing; system dynamics

PPAA '14, February 16, 2014, Orlando, Florida, USA.
Copyright © 2014 ACM 978-1-4503-2654-4/14/02…$15.00.
http://dx.doi.org/10.1145/2567634.2567637

1. Introduction

In recent years, scientific data has continued to grow in both size and complexity, arising both from large scientific experiments and from extreme scale simulations. This has been enabled by advances in instrumentation technologies that allow for highly detailed sensor readings generating large data sets, and by hardware-enabled (e.g., deep memory hierarchy, high-speed interconnect) high-fidelity simulations of complex numerical models. For example, in the fusion experiment performed in Korea, known as Korea Superconducting Tokamak Advanced Research (KSTAR) project, the data production rate is expected to be 3 TB in a 100-second experiment run when it is in full operation. Such prodigious data generation rates tax the ability to store, index, and query the results in traditional disk-based workflow approaches. As such, supporting (near) real-time data analysis offers opportunities to reduce time from experiment to scientific insight.

As an effort to deal with overwhelming data from enterprise sources, streaming-based data processing has been gaining substantial traction. Stream-based approaches have long been a part of scientific computing as well, but they have been restricted to real-time applications like haptic support within immersive visualizations. However, its capability for processing large volume data with low latency and high throughput makes it an ideal candidate for an emerging class scientific data with Near Real-Time (NRT) requirement. Specifically, this work looks to adapt streaming-based approach to the context of a new class of large experiments and simulations that have complex run-time control and analysis issues. For example, the KSTAR experimental tokamak mentioned above has been designed around the idea of analyzing and interpreting the 3TB output in time to influence the 10-15 minute setup for the next experimental run.

Additionally, users at distant locations often desire to launch *private, ad hoc* exploratory-nature online analytics during the run. If there is an interesting or abnormal phenomenon (for example those indicating instability developed in an experiment that potentially could damage experimental facilities) observed, an instant workflow needs to be injected quickly for further analysis. Simply co-locating these ad hoc tasks on-the-fly with tasks that have been already scheduled or being processed proves to be problematic and may create new contentions and pressures on the reserved resources. Our work therefore looks for more careful ways to provision and manage resources to cover both the real-time requirements and the support for ad hoc queries.

A key challenge for the types of data sources that we address is that the processing time per data element is not uniform. Generally, the processing involves the detection of features (flame fronts, plasma fluctuations), where the time to process will depend upon the number or complexity of features found in a data element, rather than just the size of the piece of data. As discussed in Section 2.2, flame front analysis for a particular set of combustion experiments utilizes two cameras, each generating an equivalent size data stream, but due to differences in experimental conditions between the two instruments the data streams may end up with substantially different processing time. As a result, standard load balancing would not be able to address the real-time scenario.

As an additional constraint, system load (CPU/IO) can vary significantly in the compute partition, and tasks that run simultaneously on a node will compete for resources [15]. As a result, some streams may get processed significantly slower than the rest. Most previous work on streaming data processing assign operators to each processing unit in a static or semi-dynamic fashion [4, 6, 13, 21, 22]. By doing worst case analysis at compilation and/or deployment, the system may be over-provisioned but has a reasonable chance to make the desired service level. However, once an operator's assignment is determined, it cannot be adjusted later on. This strategy is capable of dealing with a certain degree of load imbalance, but it struggles with use cases like those targeted in this work because of the unpredictability of compute load and system dynamics. Additionally, unlike some dynamic workloads where things are completely random, these dynamic changes have a high degree of correlation both within and between streams, since feature creation, migration, and destruction is driven by underlying physics. This work aims to develop a management system for dealing with such highly dynamic compute and analysis environments. As can be expected, this demands both light-weight continual monitoring and a very flexible load management strategy, as will be detained in later sections.

We observe that, at least for the exemplars of this class of applications which we target, that data is generated in a bursty fashion with the near real-time constraints being applied primarily between bursts, rather than within a stream. In other words, a large amount of data will be delivered within a relatively short duration, followed by an idle period, during which the data stream analysis services must be performed in order to evaluate/modify the generation of the next burst in the stream. In general, we argue that this pattern is prevalent in scientific applications, for both simulations (e.g., periodic checkpointing) and experiments, which typically produce a large amount of data outputs during each shot (i.e., in-shot), and then idle for a period of time (i.e., between-shot) during which preparation for the next shot is done.

As a specific example, in the KSTAR facility [14], after a single parameter set evaluation (shot) is finished, there is generally a period of time (typically 10-20 minutes) to analyze the data before the next experiment (referred to as *between-shot analysis*). During this time, scientists try to further analyze the data to provide direct feedback to steer the next experiment; given the distributed (cross-continent) nature of the collaborations, being able to do these streamed workflows using NRT network support could yield a substantial bonus. As another example, LAMMPS (Large Scale Atomic/Molecular Massively Parallel Simulator) [17], a molecular dynamics simulation workhorse used across a number of scientific domains, including materials science, biology, and physics to perform force and energy calculations on discrete atomic particles, typically produces data outputs in the way that every so often, e.g., after a number of user-defined epochs, it outputs the atomistic simulation data (positions, atom types, etc.), with the size of this data ranging from megabytes to terabytes depending on the science being conducted. Scientists will then apply a series of analyses on the output data to gain rapid insight like those needed to check the "health" of their running simulations [10]. The goal of this paper is, therefore, to perform as much analysis during these *between-shot* periods as is feasible, to reduce the time-to-knowledge, and to potentially save million of dollars by providing an earlier problem detection system for facilities like KSTAR where a bad parameter set could cause significant damage to the facility. Specifically, this paper makes the following contributions:

- We propose to express data streams as active data objects and embed analysis workflows in a group of related data objects for distributed data analysis. We break data outputs generated from experiments or simulations into multiple self-describing "chunks", which we call *active data objects*. This enables more flexible workflow management; scalable coordination of workflows and large-scale data streams in heterogeneous, dynamic WAN environments; and adaptive data processing to accommodate NRT processing (during an experiment/simulation run) as well as deep analysis (between the experiment/simulation runs) requirements;

- We design a novel two-tier scheme to enable an efficient active workflow management system for handling the increasing unpredictability of runtime behaviors. That is, instead of relying on determining *what* and *where* to run the scientific workflows beforehand or partial dynamically, the decision will also be adaptively enhanced online according to system runtime status. This is enabled by embedding workflow along with data streams, i.e., *active data objects*. As such, unfinished workflow pipeline can be offloaded to downstream locations for further processing.

The remainder of this paper is organized as follows. Section 2 describes two scientific applications that drive our research. Sections 3 describes the system design of the active workflow system, and the enabling techniques. Section 4 presents our preliminary experimental results. Section 5 overviews related work, followed by conclusions and future work in Section 6.

2. Motivation

In this section, we discuss some of the motivating scientific application examples for this work.

2.1 Fusion Experiments with KSTAR

KSTAR (Korea Superconducting Tokamak Advanced Research), a multi-billion-dollar fusion research device in South Korea, has been designed to study magnetic fusion reactors as a next-generation energy source [14]. The design of and experimental results from KSTAR both serve as precursor input to the ITER (International Thermonuclear Experimental Reactor) project, which is constructing the a superconducting reactor in Cadarache, France. Of concern to our work, during a typical 10-100 second fusion shot, the KSTAR device currently generates tens of gigabytes of diagnostic information that needs to be stored, analyzed and shared among scientists, ranging from basic diagnostics, such as temperature and density profiles of plasma and spectroscopies, to data sets captured by advanced imaging systems that visualize 2D or 3D structures of plasma currents, such as Soft X-ray Array (SXR), Electron Cyclotron Emissions (ECE), Beam Emission Spectroscopy (BES), etc. Furthermore, the KSTAR project has design elements requiring remote on-line collaboration, and it is planing to share experimental data with scientists even continents apart. For this purpose, during the time interval between two shots (typically 10-20 minutes), a set of data analysis tasks need to be performed remotely to analyze experiment results and

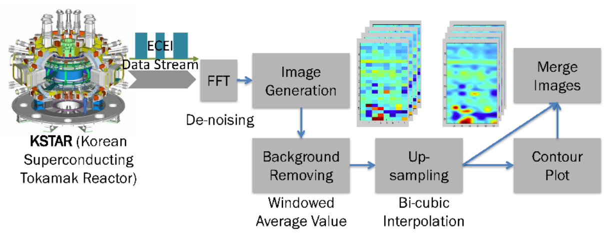


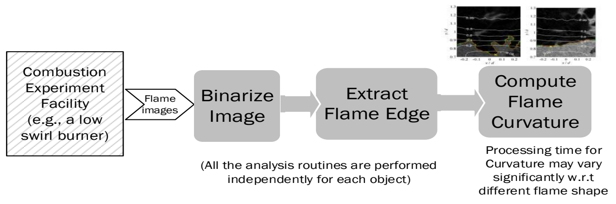
Figure 1. An example of KSTAR workflow in processing ECE images.

provide critical feedbacks for the next shot. A timely feedback can save the multi-billion-dollar device by enabling scientists to avoid using input parameter sets that might cause a dangerous disruption event. These disruptions result in a failure of the magnetic containment and cause damage to the plasma-facing wall, which is costly and time consuming to fix. An example workflow to process ECE images is shown in Figure 1.

As one may expect, performing such remote analysis through a wide-area network is a daunting task and leaves many challenges to researchers. One of the problems we are observing is that any small variation during the workflow execution, for example, due to network congestion or any straggled task execution can delay a whole workflow execution and will fail delivering timely responses. Moreover, it is difficult to make an accurate prediction on system runtime behaviors, due to the fact like system load (CPU/Network) can vary significantly and tasks that run simultaneously on a node can compete for resources and interference with each other [15]. Therefore, how to efficiently allocate computing resources for each task along the workflow execution with minimum delays is greatly desired, given the unpredictability of system runtime status. Furthermore, for the next-generation fusion reactor being built by ITER, we are expecting more powerful and longer fusion reactions which can last for 5-10 minutes. This will bring in a more serious challenge towards NRT scientific data processing.

2.2 Combustion Experiments with Low Swirl Burner

An important direction of investigation for the combustion community is to understand the physical mechanisms through which the turbulent flame speed is altered by differential diffusion effects. Changes in the turbulence of a flame can dramatically effect its stability and efficiency in engines, turbines, and other real-world applications, hence its importance. Our research team has been engaging with a group of combustion scientists. In one of their recent studies, these scientists present a new movement towards this, in which they introduce measurements to improve understandings on turbulent flame propagation characteristics of premixed, high hydrogen content (HHC) fuel/air mixtures [16]. They prove that fuel composition has virtually no effect on flame curvatures at the leading points of the turbulent flame front and that the leading point flame curvature is in-



Combustion Experiment Facility (e.g., a low swirl burner)
Flame images
Binarize Image
Extract Flame Edge
Compute Flame Curvature
(All the analysis routines are performed independently for each object)
Processing time for Curvature may vary significantly w.r.t different flame shape

Figure 2. Combustion Image Processing Workflow on Flame Curvature Detection.

stead more strongly influenced by turbulence. In order to gain this insight, the scientists conduct experiments by using a low swirl burner with a variety of H_2/CH_4 and H_2/CO mixtures at several mean flow velocities over a wide range of turbulence intensities, and then apply a customized analysis pipeline, referred to as *Flame Curvature Detection* workflow, on the collected data.

In particular, the customized workflow consists of three key analysis routines as shown in Figure 2. In order to better distinguish reactants and products, the raw image for flame tracking that is taken by experiment cameras during the experiment needs to be median-filtered; the image is then binarized with the threshold intensity selected using Otsu's method [2]; Consequently, 0 is used to represent the reactants and 1 for the products [18]. The flame edge is then extracted from the binarization of the image, in terms of $x-$ and $y-$coordinates. Finally, flame curvature, including curvature PDFs, which is useful information for examining the effects of flame stretch on the propagation rate of turbulent premixed flames is calculated at the last stage of the customized *Flame Curvature Detection* workflow.

One main challenge that the scientists encounter while processing their data according to the three-step workflow is that the processing time required by the last stage of their specified workflow, i.e., flame curvature calculation, may vary significantly with different experiment output data, referring to as the flame front tracking image in this case. This is due to the fact that some flames may be much more irregular than others as shown in the two sample flame images in Figure 2. In particular, Figure 3 displays some of the variability in the processing time for a stream of flame front tracking images output from the combustion experiment with the use of LSB. The experiment was performed with a fuel composition of 50-50 H2-CO, a swirl numer of 0.58, a mean flow velocity of 30 m/s, and an unstretched laminar flame speed of 34 cm/s. The results are normalized against the maximal processing time. As one can see in the figure, the processing time varies significantly with different experiment output data. Furthermore, the individual processing time may not be well predicted without interpreting the input 2D data. Therefore, without an efficient workflow management system, it may result in uneven computation loads among different workflow processing resources, and in the pathological case, it could cause significant delays in delivering key scientific results for the stragglers.

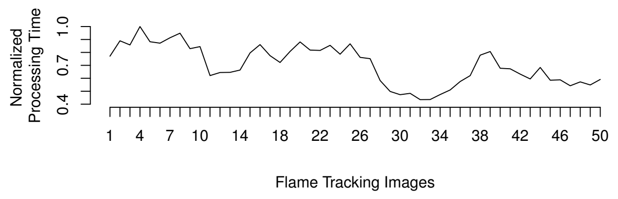
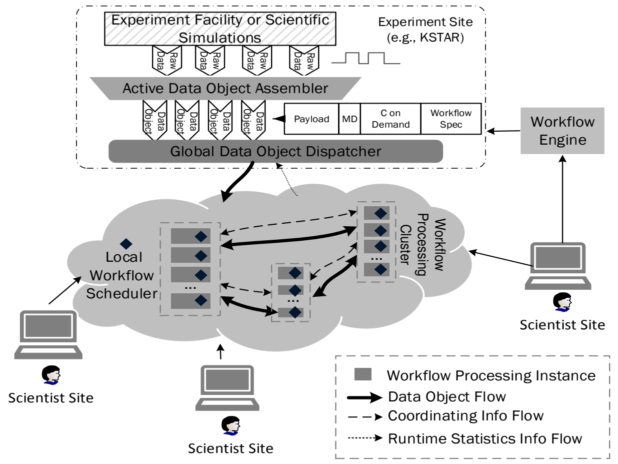


Figure 3. Processing time variation of "Flame Curvature Calculation".



Experiment Facility or Scientific Simulations
Experiment Site (e.g., KSTAR)
Raw Data
Active Data Object Assembler
Data Object
Payload
MD
C on Demand
Workflow Spec
Global Data Object Dispatcher
Workflow Engine
Local Workflow Scheduler
Workflow Processing Cluster
Scientist Site
Workflow Processing Instance
Data Object Flow
Coordinating Info Flow
Runtime Statistics Info Flow

Figure 4. Active workflow system architecture

3. System Design

In this section, we describe the proposed system architecture from a high level. The design goal of the active workflow system is to develop a runtime engine that monitors a distributed streaming environment, schedules streams and migrates streams when there is a bottleneck present, in a NRT fashion. The monitoring subsystem needs to be designed to deal with a large number of streams at distributed workflow processing resources. To realize this goal, we implement a hierarchical control framework where there is a global stream dispatcher and distributed workflow schedulers with each per workflow instance.

As shown in Figure 4, the active workflow system consists of three key components: a *stream data object assembler*, a global *data object dispatcher*, and distributed *workflow schedulers* which are located at each workflow processing instance. These components work together and orchestrate end-to-end processing. Without loss of generality, a workflow engine is also assumed to be present in the system as a gateway between scientists and the workflow system. For example, the leader of a research team may want to insert a new analysis routine to an existing workflow, given an interesting or abnormal phenomenon such as those indicating instability developed in an experiment that potentially could damage experimental facilities observed. However, in this work we mainly focus on how to devise a best-effort opportunistic workflow system according to system runtime status given the unpredictability of the system runtime behavior. Thus, we only evaluate and adopt some existing workflow techniques for the completeness of the system. Note that in terms of stream scheduling, the global stream data object dispatcher and the distributed workflow schedulers function at a different level and granularity. Namely, the stream data object dispatcher calculates a high-level and coarse-grained schedule using limited or perhaps inaccurate view of the entire system. Local workflow schedulers further adjust the schedule and migrate objects if necessary according to accurate local states, i.e., in a fine-grained scheduling fashion.

Active Data Object Assembler: It packs metadata, codes along with workflow specifications into active data objects. This makes it possible that, when a processing site receives a data object, it can decapsulate the object and understand the processing flow that is designated to it. Now in the case that an object is scheduled to be migrated, intermediate processing results are also encapsulated such that when the recipient receives the object, the workflow pipeline can restart and continue execution.

Data Object Dispatcher: It is responsible for dispatching stream data objects at data source. It obtains resource availability information from each workflow processing cluster and figures out the optimal way of processing data streams, in terms of *what* and *where* to process. In order to scale up to a large number of streams and sites, resource availability information needs to be disseminated in an efficient manner by limiting the amount of information flooded within the entire system. This can be done either by reducing the frequency of updates or by sending reduced information, which could, however, lead to a stale view of the system state in either case. This clearly requires an runtime adjustment, which is the main target of this paper.

Workflow Scheduler: A workflow scheduler resides at each processing instance and is responsible for 1) monitoring the local compute/IO load and reporting back to the global dispatcher so that the load information can be utilized to facilitate a future stream dispatch; 2) coordinating stream migration to another instance. If some workflow processing instances progress slower than others due to heavy load, part of its load can be shed and re-directed to another less loaded processing instance. This is done via closely monitoring data object queue levels, and once any queue level is below a pre-determined threshold, data object migrations are initiated. That is, at each individual workflow processing instance, we deploy a *workflow scheduler* which "watches" its local workload status. Once the local workflow scheduler observes that all the local tasks (in terms of data objects) have been provided with required analysis services, it broadcasts an "IDLE" message to all its peers within the entire workflow system; At the return of its expected feedback, it then decides which peer to "grab" tasks to provide services for.

The key enabling technique of the two-tier scheme is that we break data outputs generated from experiments or simulations into multiple self-describing "data chunks", namely, *active data objects*. Similar to the concept of data in motion [3], we then stream the data objects to distributed workflow processing resources for further science knowledge discovery. In particular, an active stream data object is defined as a data chunk associated with efficient metadata that specifies data attributes that describe the data elements captured in the data stream and their layouts within the data chunk. Also, each data element may have a time stamp, and be associated with its spatial information. Furthermore, to overcome the challenge that we may confront in the exascale computing era, that is, there might be thousands or even millions of data objects streaming in the network and thus a central workflow execution scheduler may not scale due to the synchronizations that may occur in workflows, we also embed the codes of analysis or a reference to the analysis codes, and the workflow specification information with each stream data object. Given these essential self-describing information that is embedded in stream data objects, the active workflow system gains better flexibility and opportunities for making runtime scheduling and execution decisions as a stream traverses from data source to data sink.

Additionally, the data-streaming facility in our active workflow system also requires a mechanism for enacting the flow of data objects among system-wide distributed workflow processing instances. In this work, we leverage EVPath, an event processing architecture built on Fast Flexible Serialization (FFS) which is designed to support high performance data streaming in overlay networks with internal data processing [11, 12]. In particular, EVPath enables the construction of active messaging overlay networks in the way that data stream analysis services such as data filtering and transformation functions reside in lightweight "stones" that serve as processing points in the overlay, and stones are linked to form overlay "paths". Also, the processes hosting these stones may reside on the same physical machine, on cluster nodes, or even on machines at different geographical locations, which enables the basics of adaptive, online workflow assembling in this work. Specifically, the analysis services run by stones are implemented by registered callback handlers written in C and statically associated with stones, or as inline functions deployed at runtime generated with the CoD (C-on-Demand) language. More details can be referred to in [11].

4. System Evaluation

In this section, we first give a detailed description on our experimental setup, followed by an introduction to the baseline used in this work. And then we compare the proposed active workflow system (*ActiveWorkflow*) against the baseline system (*ParDynamic*), in terms of system performance and overhead.

4.1 Experiment Setup and Baseline

Experiment Setup Due to logistic and regulation reasons, we are still in the process of getting access to real-world scientific experimental data and the associated science workflows. As a result, in this paper we are not able to conduct evaluations on the proposed active workflow system using real experimental data. However, we have been making key breakthroughs collaborating with domain scientists to move closer in this direction. In this paper, we only evaluate system performance using synthetic scientific data and workflows, which are carefully constructed to be consistent with the observations from the real-world use case. We compose the synthetic workflows closely following the *Flame Curvature Detection* use case (see Section 2.2 for more detail), since we have been closely working with combustion scientists, who have shown great interests in seeing how our system will improve combustion application performance and share with us the key parameters of their workflow.

In particular, we simulate data outputs from scientific experiments with *Data Generator* which produces a chunk of 3D data with the size of ∼2MB every 1 second during a 50 second period, which is consistent with the observation on the size of the raw flame tracking images generated by the combustion experiment as discussed in Section 2.2. Also, to be consistent with the key features of its associated workflow, we deploy a randomized algorithm to simulate the variation on the processing time occurred in the last step. Specifically, we introduce a variation according to $A + B * r$ where $r \in [0, 15]$. This is inspired by the difference in complexity that comes from the changes in the flame front. In this work, for simplification, we adopt $A = 4$ and $B = 1$ for the system evaluation. Therefore, the processing time by the last stage of the workflow will vary within the range of $[4, 19]$ seconds. Note that the parameters of the rest scheduler system may need adjustment for the specific timing of the real data, but the algorithm should hold. Additionally, we construct the synthetic workflow to perform analysis services on the output data objects as shown in Figure 5. Specifically, the synthetic workflow incorporates the key features of the *Flame Curvature Detection* image processing workflow in ways that (i) it consists of three analysis routines; (ii) each analysis routine performs service on each data object independently; (iii) only the last analysis routine of the 3-step workflow demonstrates a variation on its processing time with different experiment data outputs. Note that the specific analysis routines that are used to compose the synthetic workflow, including *Derivative*, *Second Derivative*, and *Local Min&Max*, are designed in the way to ease the synthetic workflow construction, as well as to be meaningful for the simulated data outputs. However, any other workflow with the three key features as described above can be easily adopted for the system evaluation.

Specifically, in the following measurements, we fixed the ratio of the total number of compute cores used by workflow processing instances to data generator to 0.2. Also, in order to simulate a cloud of resources providing workflow services that are distributed across networks, we assume that there are two independent clusters with each instantiating half of the total workflow processing instances. Accordingly, two independent clusters of workflow processing instances with each having 2 processing instances will be deployed into the system. Lastly, the proposed active workflow system is evaluated using the Sith cluster hosted at Oak Ridge National Laboratory, which consists of 40 compute nodes and each node is equipped with four 2.3 GHz 8 core AMD Opteron processors and 64 GB of memory.

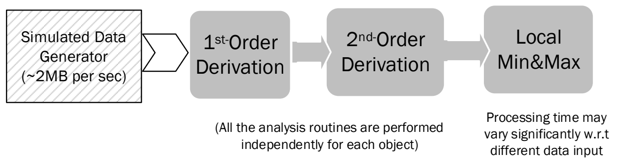

Simulated Data Generator (~2MB per sec)
1st-Order Derivation
2nd-Order Derivation
Local Min&Max
(All the analysis routines are performed independently for each object)
Processing time may vary significantly w.r.t different data input

Figure 5. Synthetic workflow for system evaluation.

Baseline In this paper, we introduce one primary baseline for comparison purpose, referred to as *ParDynamic*. Similar to *ActiveWorkflow*, the baseline system tries to predict a best distribution strategy for distributing output data that is streaming out from Data Generator to the pool of distributed workflow processing instances at the end of an experimental shot. *ParDynamic* is similar to the state-of-the-art work [4, 6, 13, 21] in the sense that the scheduler determines the stream assignment during runtime according to the latest system-wide state. However, as opposed to *ActiveWorkflow*, the baseline system is not aware of system runtime variations.

4.2 Impacts on Workflow Completion Time

In this experiment, we evaluate the effects on the near real-time extreme-scale science in terms of workflow completion time, by the active workflow management system. In particular, we use weak scaling to show how the system behaves in terms of different scales of data generator participants.

Figure 6 demonstrates the entire workflow completion time by *ActiveWorkflow*, and *ParDynamic*, respectively, with respect to a series of different scales of data generator participants. As shown in the figure, compared to the baseline system, the active workflow system can speed up the workflow completion time by up to 27.7%. This is achieved by the proactive role of the distributed workflow scheduler component.

In particular, Figures 7 and 8 give a detailed description on the corresponding dynamics of the tasks (in terms of semi-processed data objects) for the last analysis routine whose processing time varies significantly regarding different experiment output data, with the experiment configuration of 20 data generator participants. We refer to a specific instance in the form of (Ri, Ij), where i and j represent the specific cluster, and instance, respectively. As shown in Figure 7, instances $(R1, I1)$ and $(R1, I2)$ can be identified as "swift" workflow processing entities, since they retire their

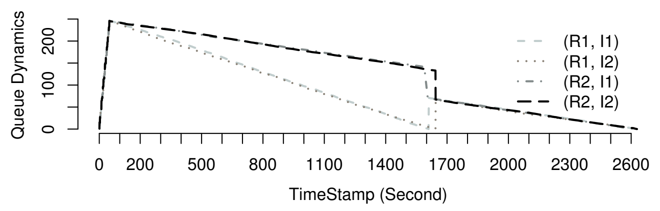


Figure 7. Queue dynamics (in terms of semi-processed data objects) for Stage#3 by *ActiveWorkflow*, w.r.t the scale of 20 Data Generator processes.

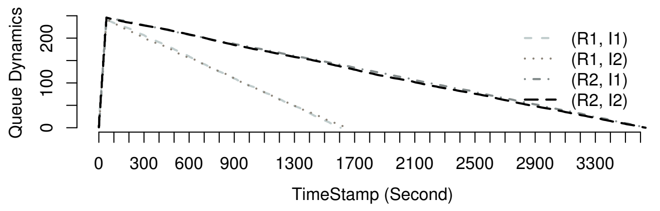


Figure 8. Queue dynamics (in terms of semi-processed data objects) for Stage#3 by *ParDynamic*, w.r.t the scale of 20 Data Generator processes.

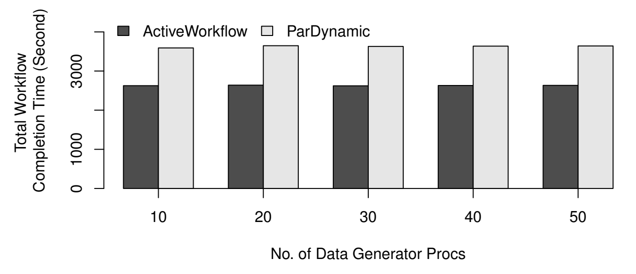


Figure 6. Workflow completion time by *ActiveWorkflow*, and *ParDynamic*, respectively.

data objects in a relatively faster fashion, i.e., the decreasing slope of the queue dynamics is steeper, compared to their "clumsy" peers of ($R2$, $I1$) and ($R2$, $I2$). One can observe from the figure that two major load shifting actions occur in the system: (i) when the instance ($R1$, $I1$) turns into idle after servicing all its streams; and (ii) when the instance ($R1$, $I2$) turns into idle. Accordingly, the instance ($R2$, $I1$) sheds half of its tasks to the instance ($R1$, $I1$) at the first load shifting. This can be observed from the fact that the decreasing slope of the corresponding "clumsy" instance experiences a sharp change near the timestamp of 1700 since the workflow system starts and then return to its regular "retiring" speed. Therefore, as promised by the active workflow system, once there is a transient hotspot observed in the system like those occur at the time of un-even computation load distribution, a load shifting is accordingly performed by breaking the previous schedule of the stream and searching for a least loaded location to continue the execution of the remaining workflow steps. In contrast, the baseline system does not introduce any exchange of the semi-processed tasks among different workflow processing instances once the system starts as shown in Figure 8, due to its unawareness of the system runtime behavior.

4.3 System Overhead

In this experiment, we evaluate the system overhead in terms of "extra" data traffic caused by the active load shifting with the use of the active workflow system, with respect to a series of different scales of data generator participants. In particular, we refer to $\frac{TotalDataMovement}{TotalDataGeneratorOutput}$ as the extra data traffic for the system.

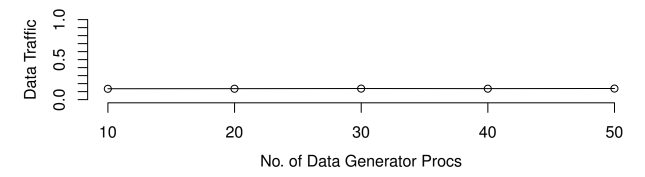


Figure 9. Extra data traffic by *ActiveWorkflow*.

As shown in Figure 9, the extra data traffic introduced into the system by the active load shifting is reasonably scalable with the increase of data generator participants in the context of weak scaling. Also, due to the broadcast-based message exchanging protocol used by the workflow scheduler for active load balancing, each data movement is preceded with $2 * N - 1$ light-weight control messages, where N is the total number of workflow processing instances that are deployed in the system. Next, we are planning to devise a more intelligent strategy for the distributed workflow schedulers, trading off between a better utilization of workflow processing resource and the overhead of data movement given the current network resource status, in order to achieve a best-effort active workflow management system in our future work.

5. Related Work

In the last several years, extensive research has been done towards developing distributed data stream processing systems from both academics and industries [1, 4–7, 9, 20]. Due to lack of space, however, we mainly discuss the pieces that are particularly relevant to our work and highlight the primary differences.

As one may expect, one key component of any data stream processing system is a scheduler that promises a best-effort resource allocation. Wolf et al propose an optimizing scheduler for the System S [21], referred to as SODA, which determines a global placement and scheduling plan to advise processing element executions on processing nodes, according to the collected runtime statistics by its resource manager. Different from the active workflow system proposed in this work, the optimization-based scheduler is to maximize a utility-theoretic function on the "importance" measured at the terminal streams of the data flow graphs, with the assumption that it has a good knowledge on the resource requirement of each processing element, as well as the system runtime status. While in this work, we acknowledge that it is not entirely possible to pre-determine how much resources (e.g., CPU) need to be allocated at each processing node for exploratory-nature analytics. Also, system load (CPU/IO) can vary significantly and tasks that run simultaneously on a node can compete for resources and interference with each other.

In another recent study [13], Khandekar et al. propose to leverage a minimum-ratio cut-based algorithm to optimally fuse compile-time operators into processing elements, in order to maximize job throughput by minimizing the processing cost associated with inter-processing elements stream traffic. In particular, the specific optimizer relies on application-level information such as performance metrics indicating the CPU demands of the operators and data rates of each stream in the system. This violates the observations as discussed in Section 1, i.e., a reasonable prediction regarding stream processing time may not be feasible [16]. Schneider et al. [19] devise an adaptive algorithm to scale the performance of data analytics operators for stream processing in response to changes in incoming workload and the availability of processing cycles, by adjusting the level of parallelism at runtime such as creating new worker threads or putting workers into sleep. Instead, the active workflow system is built upon the practice that the level of parallelism is fixed once the system starts due to the fact that with current high end machines, batch schedulers typically assign to each user the amount of requested resource for the entire duration of their applications execution.

Aurora project [4] is another recent effort that aims to develop a single infrastructure that can process continuous *push-based* data streams efficiently. It looked at many research challenges, such as query scheduling, runtime stream management to deal with transient load spikes, failure detection and recovery, from the new pushed-based paradigm. Carney et al. [8] propose a two-level scheduler to be used in Aurora for a dynamic scheduling-plan construction, in terms of which operators to schedule, in which order to schedule the operators, and how many tuples to process at each operator execution. Also, a load shedder will be activated once an overload situation is detected. In contrast to this work, our active workflow system propose to break data from data source into multiple active data objects which contain self-describing information like analysis specification. Thus, a stream is free to break its previous schedule at any time for a better system performance, if necessary.

Another recent paper that comes out from Aurora project which is very relevant to this paper is the dynamic load distribution strategy [22]. By and large, the goal of query optimization is to map operators efficiently to resources in a distributed environment. However, due to the load dynamics, this mapping will need to adapt, thereby avoiding hotspots in the system. Xing at al, developed a correlation based load distribution algorithm that avoids load spikes and minimizes latency by minimizing load variance. However, once operators are mapped to resources, a stream coming in is fixed in terms where to go and what to be processed in its lifetime. This work, in contrast, goes one step ahead and develop an elastic system that allow a stream to change its schedule while it's being processed.

6. Conclusion and Future Work

Given the observations on the increasing unpredictability of compute load and system dynamics such as data element processing time may not be reasonably predicated [16] and system load (CPU/IO) can vary significantly in the compute partition because tasks that run simultaneously on a node will compete for resources [15], this work proposes an active workflow management system for dealing such highly dynamic compute and analysis environments. In particular, we deploy a novel two-tier scheme for handling the increasing unpredictability of runtime behaviors: Instead of relying on determining *what* and *where* to run the scientific workflows beforehand or partial dynamically, the decision will also be adaptively enhanced online according to system runtime status. This is enabled by embedding workflow along with data streams. Specifically, we break data outputs generated from experiments or simulations into multiple self-describing "chunks", which we call *active data objects*. As such, if there is a transient hotspot observed, a data object with unfinished workflow pipeline can break its previous schedule and search for a least loaded location to continue the execution. Our preliminary experiment results demonstrate the proposed active workflow system as a very promising solution by outperforming the state-of-the-art semi-dynamic workflow schedulers with an improved workflow completion time, as well as a good scalability.

As an initial effort, this work mainly discusses the design of the distributed *workflow scheduler* component of the active workflow management system, given the unpredictability of compute load and system dynamics. In particular, a threshold-based strategy is used to advise the runtime adjustment as detailed in Section 3. Next, we are planning to deploy a more intelligent strategy for the distributed workflow schedulers, to best tradeoff between workflow processing resource utilization and extra data movement traffic introduced into the system given the current network resource status. Any online adjustment comes at a price, even if it may be a light weight method. Therefore, we also want to investigate how to best devise the global dispatcher, to make the local workflow schedulers invoked as infrequent as possible, in order to reduce overhead due to active runtime adjustments.

7. Acknowledgements

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. Additionally, this work was supported in part by the Department of Energy Office of Advanced Scientific Computing Research and by the National Science Foundation.

References

[1] Streambase systems. `http://www.streambase.com`.

[2] A Threshold Selection Method from Gray-Level Histograms. *IEEE Trans. Syst. Man Cybern*, 9(1):62–66, 1979.

[3] Harnessing Data in Motion. Technical report, IBM, 2010.

[4] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A New Model and Architecture for Data Stream Management. *The VLDB Journal*, 12(2):120–139, Aug. 2003.

[5] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. hyon Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the borealis stream processing engine. In *CIDR*, 2005.

[6] L. Amini, H. Andrade, R. Bhagwan, F. Eskesen, R. King, Y. Park, and C. Venkatramani. SPC: A distributed, scalable platform for data mining. In *DM-SSP*, 2006.

[7] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, K. Ito, R. Motwani, U. Srivastava, and J. Widom. STREAM: The stanford data stream management system. Springer, 2004.

[8] B. Babcock, S. Babu, M. Datar, R. Motwani, and D. Thomas. Operator Scheduling in Data Stream Systems. *The VLDB Journal*, 2004.

[9] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous Dataflow Processing. In *SIGMOD*, 2003.

[10] J. Dayal, J. Cao, G. Eisenhauer, K. Schwan, M. Wolf, F. Zheng, H. Abbasi, S. Klasky, N. Podhorszki, and J. Lofstead. I/O Containers: Managing the Data Analytics and Visualization Pipelines of High End Codes. In *IPDPSW*, 2013.

[11] G. Eisenhauer, M. Wolf, H. Abbasi, and K. Schwan. Event-based systems: opportunities and challenges at exascale. In *DEBS*, 2009.

[12] G. Eisenhauer, M. Wolf, H. Abbasi, S. Klasky, and K. Schwan. A Type System for High Performance Communication and Computation. In *eScienceW*, 2011.

[13] R. Khandekar, K. Hildrum, S. Parekh, D. Rajan, J. Wolf, K.-L. Wu, H. Andrade, and B. Gedik. COLA: Optimizing Stream Processing Applications via Graph Partitioning. In *Middleware*, 2009.

[14] G. Lee et al. Design and construction of the KSTAR tokamak. *Nuclear Fusion*, 41(10):1515, 2001.

[15] Q. Liu, N. Podhorszki, J. Logan, and S. Klasky. Runtime I/O Re-Routing + Throttling on HPC Storage. HotStorage, 2013.

[16] A. Marshall, P. Venkateswaran, J. Seitzman, and T. Lieuwen. Measurements of Leading Point Conditioned Statistics of High Hydrogen Content Fuels. In *The 8th U.S. National Combustion Meeting*, 2013.

[17] S. Plimpton, R. Pollock, and M. Stevens. Particle-Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations. In *PPSC*. SIAM, 1997. ISBN 0-89871-395-1.

[18] T. Poinsot and D. Veynante. *Theoretical and Numerical Combustion*. R.T. Edwards.

[19] S. Schneider, H. Andrade, B. Gedik, A. Biem, and K.-L. Wu. Elastic scaling of data parallel operators in stream processing. In *IPDPS*, 2009.

[20] W. Thies, M. Karczmarek, and S. P. Amarasinghe. StreamIt: A Language for Streaming Applications. In *CC*, 2002.

[21] J. Wolf, N. Bansal, K. Hildrum, S. Parekh, D. Rajan, R. Wagle, K.-L. Wu, and L. Fleischer. SODA: An Optimizing Scheduler for Large-scale Stream-based Distributed Computer Systems. In *Middleware*, 2008.

[22] Y. Xing. *Load Management Techniques for Distributed Stream Processing*. PhD thesis, 2006.