PLAStiCC: Predictive Look-Ahead Scheduling for Continuous dataflows on Clouds

Alok Kumbhare[1], Yogesh Simmhan[2] and Viktor K. Prasanna[1]
[1]*University of Southern California, Los Angeles, California 90089*
[2]*Indian Institute of Science, Bangalore 560012*
Email: kumbhare@usc.edu, simmhan@serc.iisc.in, prasanna@usc.edu

***Abstract*—Scalable stream processing and continuous dataflow systems are gaining traction with the rise of big data due to the need for processing high velocity data in near real time. Unlike batch processing systems such as MapReduce and workflows, static scheduling strategies fall short for continuous dataflows due to the variations in the input data rates and the need for sustained throughput. The elastic resource provisioning of cloud infrastructure is valuable to meet the changing resource needs of such continuous applications. However, multi-tenant cloud resources introduce yet another dimension of performance variability that impacts the application's throughput. In this paper we propose *PLAStiCC*, an adaptive scheduling algorithm that balances resource cost and application throughput using a prediction-based look-ahead approach. It not only addresses variations in the input data rates but also the underlying cloud infrastructure. In addition, we also propose several simpler static scheduling heuristics that operate in the absence of accurate performance prediction model. These static and adaptive heuristics are evaluated through extensive simulations using performance traces obtained from public and private IaaS clouds. Our results show an improvement of upto 20% in the overall profit as compared to the reactive adaptation algorithm.**

***Keywords*-Continuous Dataflows; Predictive scheduling; IaaS Clouds; Elastic resource management; Stream processing**

I. Introduction

There has been a tremendous rise in the capability to collect data – pervasively and continuously – for a wide range of application domains such as financial trading, social networks, and cyber-physical systems (CPS) like Smart Power Grids and Smart Transportation. Consequently, there is increasing attention on data analytics frameworks that perform continuous data processing with low latency and guaranteed throughput. Batch processing systems [1], [2] that deal with high volumes of slow changing data abound. In contrast, continuous applications are characterized by data streaming with high velocity and at variable rates, on which multiple stages of information integration and analytics are performed. They offer online insight to domain experts (or their digital agents) to help detect and prevent frauds, power outages or security situations, as may be the case. Runtime environments for such applications demand *continuous adaptation of user logic and dynamic scaling of resources, under changing stream workloads*, to meet the user's quality of service (QoS) requirements.

Cloud computing platforms, especially Infrastructure-as-a-Service (IaaS), provide flexible, on-demand resources. Their elastic resource provisioning is well suited to scale resources at runtime in response to changing needs of continuous applications. Several stream processing frameworks [3], [4], [5] and adaptive provisioning algorithms [6], including our own earlier work [7], [8], utilize this online elasticity to balance resource cost against application performance in response to changing data rates.

However, cloud infrastructure exhibit *performance variability* for virtualized resources (e.g. CPU, disk, network) as well as services (e.g. NoSQL store, message queues) both over time and space. These variations may be caused by factors including shared resources and multi-tenancy, changing workloads in the data center, as well as diversity and placement of commodity hardware [9]. For time sensitive continuous dataflow applications, it becomes imperative to address such fluctuations in the performance of virtualized Cloud resources to ensure that the desired QoS is maintained. Strategies may range from simple over-provisioning and replication to dynamic application re-composition [8] and preemptive migration [10].

A pro-active management of application and resource mapping is possible with predictive models that can forecast resource behavior. Several performance models have been proposed for computational Grid infrastructure that also exhibit performance variability [11]. These models use the network topology and workload characterization to estimate the behavior of specific Grid resources [12], [13]. Similar models have also been developed for short and medium term predictions of performance metrics for virtual machine (VM) clusters using machine learning techniques such as discrete-time Markov chain [14]. Hence, it is feasible to ascertain the future performance of Cloud resources, to different degrees of confidence.

Given the ability to elastically control Cloud resources and the existence of resource performance prediction models, the question arises: *How can we replan the allocation of elastic resources for dynamic, continuous applications, at runtime, to mitigate the impact of resource and workload variability on the QoS of the application?*

In answering this question, we consider continuous dataflows as the abstraction used to compose these continuous applications. In our previous work [8], we considered proposed adaptive scheduling heuristics for online task re-composition of dynamic and continuous dataflows as well as allocation and scaling of cloud resources. However, these strategies *reacted* to changes in workloads and resource variations to ensure QoS targets were met. In this paper, we leverage the knowledge of future resource and workload behavior to *pro-actively* plan resource allocation, and thus limit costly and frequent runtime adaptation. We propose a new Predictive Look-Ahead Scheduling algorithm for Continuous dataflows on Clouds (*PLAStiCC*) that uses short term workload and infrastructure performance predictions to actively manage resource mapping for continuous dataflows to meet QoS goals while limiting resource costs. We also offer, as alternatives, simpler scheduling heuristics that enhance the reactive strategy from our prior work. The specific contributions of the paper are as follows:

1) We analyze performance traces for public and private Cloud VMs to highlight spatio-temporal variations (§II). These motivate our problem while offering insights to build resource forecasting models; however, the modeling itself is beyond the scope of this paper.
2) We build upon our prior continuous dataflow abstraction (§III) to formalize their scheduling on Clouds VMs as an optimization problem (§IV), the solution for which will give an optimal mapping of tasks to cloud resources.
3) We propose several scheduling heuristics for elastic resources, including the novel PLAStiCC algorithm, that leverage short and medium term performance predictions (§V).
4) Finally, we evaluate the proposed heuristics through extensive simulations that use real performance traces from public and private Cloud VMs for several continuous dataflow applications and workloads (VI).

II. Analysis of Cloud Performance Variability

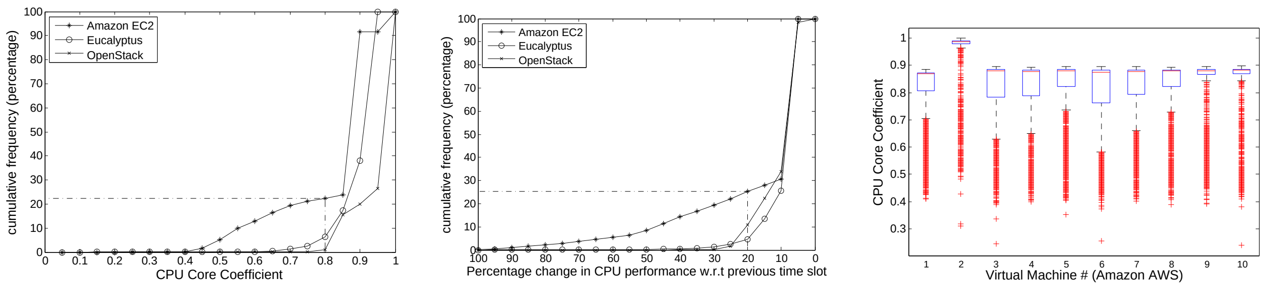
Performance variations present in virtualized and multi-tenant infrastructures like Clouds make it challenging to run time-sensitive applications such as continuous dataflows. Several studies have observed such variations both within a VM and across VMs [9]. Their cause is attributed to factors including multi-tenancy, evolving hardware generations in the data center, placement of VMs on the physical infrastructure, hardware and software maintenance cycles, and so on. In this section, further empirical evidence of such performance variation, both on public and on private clouds, are offered. This analysis motivates the need for our proposed predictive look-ahead scheduling algorithm and while offering insights on building performance prediction models used by our algorithm. Developing the actual performance prediction models are beyond the scope of this paper.

Two Cloud providers are considered: *FutureGrid*, a private academic IaaS Cloud running *Eucalyptus 2.0* and *OpenStack Grizzly (v7.0)* Cloud fabrics, supporting over 150 Cloud research projects [1], and *Amazon EC2*, a popular public commercial IaaS Cloud. The experimental setup consists of running three groups of VMs: 20 Eucalyptus, and 10 OpenStack VMs on FutureGrid, and 10 EC2 VMs on Amazon, over 1–26 day period. Each VM's class was equivalent to Amazon's `m1.small`, with one dedicated virtual CPU core and 1024 MB of memory. Standard CPU (whetstone[15]), disk, and memory benchmarks are run on each VM every 2 mins, and a peer-to-peer network benchmark between the VMs measure TCP connection time, packet latency and bandwidth ([16]). Here, we focus only on the CPU and network performance metrics since our proposed algorithms only account for these. However, this can be extended to consider variations in metrics like local disk I/O, durable storage and messaging services (Amazon S3, SQS), etc. based on the application's characteristics.

We are interested in the deviations of the VMs from their *rated* (i.e. ideal) performance rather than their absolute performance. Hence, for each of the three VM groups, each observed measurement for a performance metric is normalized based on the best observed performance for that metric, i.e., we divide each measurement by the best seen value in the VM group, which is the largest for bandwidth, and smallest time for CPU, network connect and latency. We also take the reciprocal for metrics where smaller is better so that in these plots, 1.0 always refers to the best seen value.

Figure 1(a) shows the cumulative distribution function (CDF) for the normalized CPU core performance (X axis) across all VMs and across time, in each of the three VM groups. The Y axis shows the cumulative fraction of the number of VMs whose observed performance is less than the normalized performance on the X axis. We see that all three VM groups and service providers show variations in performance, with the EC2 VMs on Amazon's public Cloud showing the highest variations, possibly due to greater data center diversity and high multi-tenancy and utilization of physical resources. We also notice that many VMs fail to reach the ideal 1.0 normalized performance. 23% of EC2 VMs have a normalized core performance of < 0.80, while $< 10\%$ of the FutureGrid VMs exhibit this shortfall, possibly due to more uniform physical resources. *A wide divergence in rated and available computational capability can tangibly impact the overall application QoS* (§ VI). Variations are also seen in the network performance measures when considering all VMs over time; we do not plot this in the interests of space. In brief, we see a high deviation in packet latency for Amazon EC2, with more than 45% of VMs exhibiting a normalized packet latency greater than 0.70; the network bandwidth for 22% of EC2 VMs have a bandwidth coefficient < 0.80). These can punitively impact the exchange of small messages by streaming applications. These performance variations in CPU and network characteristics

[1]https://portal.futuregrid.org/projects-statistics



(a) Cumulative Distribution Function (CDF) of normalized CPU Core performance, across VMs and over time.

(b) CDF of performance variation relative to previous measurement time-slot.

(c) Box-and-Whiskers plot of normalized performance distribution over time (Y axis) and for different Amazon AWS VMs (X axis).

Figure 1. CPU Core performance characteristics, across space and time.

cause the application's QoS to degrade and demonstrate the need for dynamic runtime adaptation.

Our prior work on dynamic application adaptations [8] accounted for the diversity in performance across VMs and small/slow changes in performance using reactive (after-the-fact) strategies. However, we also observe transient and sharp fluctuations in performance (i.e. high amplitude of Δ, small duration), both in CPU as well as network performance, within the same VM. Figure 1(b) shows the percentage change (unsigned) in average normalized CPU performance between consecutive 5 minutes measurement periods. Larger the % value of the X Axis (Left), greater the variation in performance between consecutive periods, and more pronounced is the short term performance fluctuations. In about 25% of the observations, the performance changes by $> 20\%$ within a 5 min period for an EC2 VM. The extent of variations is also different for different VMs. Figure 1(c) shows a box-and-whiskers plot on the Y axis representing the average normalized CPU performance for each 5 min interval and the X axis representing different EC2 VMs. The height of the box shows the difference in performance between q_1 and q_3 quartiles (i.e. 25^{th} and 75^{th} percentiles). Also, a number of outliers that fall below ($< q_1 - 1.5 \times (q_3 - q_1)$) are also observed. Figure 2 shows the temporal deviations between successive 5 min time slots for the same VM for the network characteristics. Here too we see that for $20 - 30\%$ of the cases, there is a $> 20\%$ change in network performance, depending on the metric. A detailed discussion is omitted for brevity.

Such performance fluctuations and outliers mean that the immediate past performance of a resource may not be sustained in the immediate future, and assuming so will cause sub-optimal resource allocation strategies. In particular, such transient performance changes can cause decisions that are unrolled soon after. A look ahead adaptation strategy that not only accounts for the performance variations across VMs, but is also resilient to temporal changes in the short and long terms within a VM is necessary.

III. Preliminaries & Background

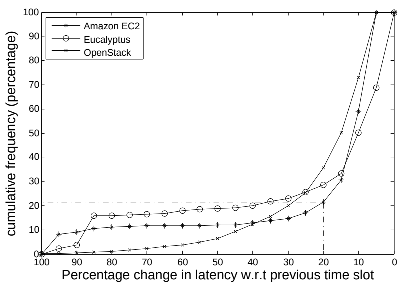
We introduce the abstraction of continuous and dynamic dataflows, and summarize our existing work on reactive scheduling of such dataflows on Clouds with infrastructure variability [8]. We build upon this for our current contribution on pro-active scheduling with look ahead.

A *continuous dataflow*, also known as a stream processing application, is a directed acyclic task graph (DAG), $G = (P, E, I, O)$, where $P = \{P_i\}$ is the set of *processing elements (PEs)*; $E = \{e_{i,j} : \exists$ a dataflow edge between P_i and $P_j\}$ is a set of dataflow *channels* between the PEs; and $I \neq \varnothing \subset P$ and $O \neq \varnothing \subset P$ are the sets of input PEs (without an incident channel) and output PEs (without an outgoing channel), respectively. Each PE is a long-running user-defined compute logic, accepting and processing data messages on the incoming channels and producing data messages on the outgoing channels. This DAG-based application composition model, prevalent in various domains, provides a flexible abstraction to build complex continuous dataflow applications [17], [18].

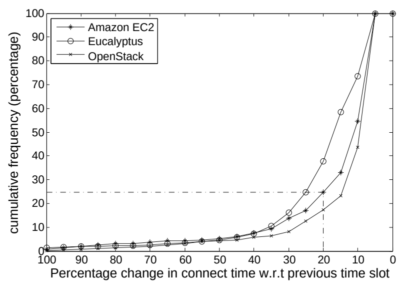
A. Dynamic Continuous Dataflows

Continuous dataflows are extended to dynamic dataflows that allow run-time application re-composition in response to the domain triggers. They also provide an additional dimension of control to schedule the dataflow to balance application QoS, resource cost and application value.

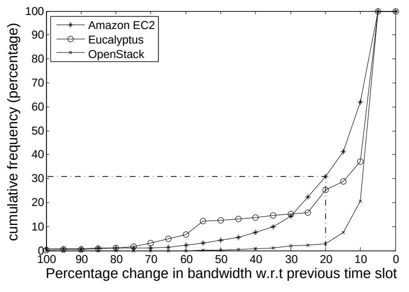
The main characteristic of a *dynamic dataflow* is the set of "alternates" [19] available for individual PEs. This lets the user define more than one implementation (alternates) for each PE. All alternates for a PE perform the same logical operation but with a different trade-off between the domain perceived "value" and the resource needs. Only one alternate is active for a PE, and the choice of activating an alternate does not impact the dataflow correctness, nor does changing the active alternates while the continuous dataflow is executing. The latter allows the framework to schedule any alternate for a PE without affecting application correctness, just quality. Each alternate p_i^j of a PE (P_i) is associated with the QoS, resource and execution metrics: relative domain value, γ_i^j, resource cost, c_i^j, and selectivity, s_j^i.



(a) Latency



(b) Connect



(c) Bandwidth

Figure 2. CDF of variation in normalized network performance measures relative to previous measurement time-slot.

The *relative domain value*, $0 < \left(\gamma_i^j = \frac{f(p_i^j)}{\max_j f(p_i^j)}\right) \le 1$, is the domain perceived relative benefit of activating that alternate for a given PE relative to its best available alternate, where $f(p_i^j)$ is a user-defined positive valued function, e.g., the F_1 statistical measure. In a timestep, t, the *application value* for the entire dataflow is the aggregation of all active alternates' values, one for each PE, in that duration.

The *resource cost*, c_i^j, is the number of core-seconds required to process a single incoming message by an alternate on a "standard" CPU core. This restricts the PE's internal logic to use a single-core implementation and a *standard CPU core* is an infrastructure specific *benchmarked* (as opposed to rated) unit of computing in a heterogeneous Cloud environment. These allow us to use a simple linear scaling to obtain the processing requirements for that alternate on other benchmarked CPUs. For e.g., given an alternate that requires 3.6 secs to process a single message on a standard 1 Ghz core, the time required to process that alternate on another core with processing capacity benchmarked at 1.8 Ghz can be calculated as, $processing\_time_x = \frac{processing\_time_{std}}{processing\_capacity_x} = \frac{3.6}{1.8} = 2secs$.

Selectivity, s_i^j, is the ratio of the number of messages *produced* by the alternate to the number of messages *consumed* by it to complete a logical unit of operation by that alternate. Selectivity determines the message load on the downstream PEs in the dataflow. The impact of an alternate on overall dataflow cost is thus its own resource cost and its contribution to the cost of downstream PEs.

The dataflow application is deployed in a distributed environment [20]. One alternate is active for each PE at deployment and the scheduler can switch between alternates during the application's runtime. Several *instances* of an alternate can run in parallel across independent cores; while instances may inter-communicate through some persistence mechanism, for simplicity, we assume that each instance of a PE is data parallel and can execute independently. So, a PE can be scaled up/down by increasing/decreasing the number of instances, each using an *exclusive* core. [8] illustrates such dataflows, alternates, and their resource mapping to VMs.

The QoS of the application is measured by the *relative application throughput*, Ω, which is the ratio of observed output throughput to the maximum achievable output throughput, given the incoming data rate to the dataflow. The observed throughput depends on the number of instances of each PE and the resources assigned to them. The dataflow's lifetime is divided into *optimization periods* of duration T, and the user specifies as a QoS constraint the *minimum average* relative throughput, $\widehat{\Omega}$, to be met for each T.

The *overall profit*, Θ, for the application deployment is defined as $\Theta = \Gamma - \sigma \times \mu$, where $0 \le \Gamma \le 1$ is the normalized application value and μ is the dollar cost incurred by the application, both defined over each optimization period T, and σ is the *value coefficient* defined as:

$$\sigma = \frac{\text{Max Application Value} - \text{Min Application Value}}{\text{Acceptable Cost at MaxVal} - \text{Acceptable Cost at MinVal}}$$

The optimization goal for scheduling dynamic continuous dataflows, thus, is to maximize the overall profit, Θ, while meeting the user defined QoS constraint $\Omega \ge \widehat{\Omega}$ for each optimization period.

B. Reactive Runtime Scheduling and Resource Mapping

To achieve the optimization goal, the scheduler can, at runtime: (1) switch the alternate for a PE, (2) change the number of instances for an alternate, and (3) select the Cloud VMs on which instances are deployed. These can be in response (i.e. reactive) to changes in the incoming data rates and variability in infrastructure behavior (or even the scheduler's prior sub-optimal decision), that cause a deviation from the goal. Our prior work on a reactive scheduling algorithm [8], discussed in brief here, forms the basis for the proposed PLAStiCC algorithm (§ V).

The scheduling algorithm has two phases [8]. In the *initial deployment phase*, the algorithm assumes that the rated performance for the Cloud resources and estimated data rate suggested by the user are accurate, and allocates resources to each PE. This determines the initial alternates that are active, the number of data parallel instances of each, the number and size of VMs to be launched, and the mapping of PE instances to these VMs. In the *runtime adaptation phase*, the scheduler continuously monitors the dataflow and infrastructure, measuring the current relative throughput, incoming data rates, and the CPU performance for, and bandwidth between, different VMs. Based on these *observed* performance metrics, it can decide to change the active alternates, and scale up/down the number of VMs and PE instances to balance the QoS against resource cost.

The reactive algorithm [8] has two independent stages, *alternate re-deployment* and *resource re-deployment*. The former selects feasible alternates for each PE such that they do not require any changes to the resources required. This depends not only on the active alternates but also on the current resource performance. It then picks the alternate for a PE with the highest value relative to its cumulative cost, i.e., the sum of its own resource needs and resources needed by downstream PEs due to its selectivity. Resource re-deployment, on the other hand, preserves the active alternates while it scales up/down the number of instances of the alternates and launches/releases VMs. Here, the heuristic makes several decisions: *Which PE is under/over provisioned?*, *Which PE instance has to be scaled down?*, and *Which VM has to be shutdown?*. These decisions affect the overall application QoS, its value, as well as the resource cost. [8] offers a comparison of these heuristics for runtime adaptation.

The scheduler uses the monitored application and infrastructure metrics to decide on initiating alternate or resource re-deployment. The "reactiveness" is because the scheduler's decisions assume that the observed data rates and resource performance from the immediate past will sustain in the future. Although this reactive algorithm mitigates the effects of variations in workload and infrastructure performance, due to its myopic nature, it is susceptible to spikes and short term fluctuations in the VMs' performance. This leads to adaptation decisions (e.g. launching a new VM in response to a transient performance drop) that need to be reversed in the near future (e.g. once the VM's performance is regained). Besides adding to the scheduling overhead, it also increases the cost of Cloud resources that are billed by the hour.

IV. Problem Formulation

Reactive scheduling can be mapped to the constrained utility maximization problem applied to continuous dataflows [8]. We define the pro-active *look-ahead optimization* problem as a refinement of this by including predictions for the incoming data rates and performance of the underlying infrastructure. In contrast, the reactive problem only has access to the current snapshot of the behavior.

We define a fixed optimization period T over which the application schedule is to be optimized for while satisfying the QoS constraint. T may be the duration of the application lifetime or some large time duration. T is divided into a number of fixed length timesteps, $t_0, t_1, ..., t_n$. The initial deployment decision, based on the rated infrastructure performance and user estimated data rates, is done at the start of timestep t_0, the system is monitored during t_i and runtime decisions performed at the start of timestep t_{i+1}.

Figure 3 (a) shows the *reactive optimization problem* defined over an optimization period T. The profit (i.e. utility) function to maximize, Θ, is defined over the normalized application value and the total resource cost over the entire period T, given by $\Theta = \Gamma - \sigma \times \mu$. Similarly, the application throughput constraint $\Omega \geq \widehat{\Omega}$ is defined over the average throughput observed over T. Consequently, the instantaneous Ω at a given time may not satisfy the given constraint though the average is satisfied by the end of T.

This problem is modified for pro-active *look-ahead optimization* as follows. Since prediction algorithms are often accuracy over short horizons (i.e. near future), and to reduce the time complexity of the problem, we reduce the above global optimization to a set of smaller local optimization problems (Figure 3 (b)). First, the optimization period T is divided into *prediction intervals* $[P_0, ..., P_k]$, each of size equal to the length of the prediction horizon given by the prediction model ("oracle"). As before, we divide each interval P_j into timesteps $t^j_0, t^j_1, ..., t^j_k$, at the start of which the adaptation decisions are made.

Next, a stricter version of the application constraint is defined, requiring it be satisfied for each prediction interval P_j, i.e. $\forall P_j : \Omega^p_j \geq \widehat{\Omega}$. It is evident that if this constraint is satisfied for each interval, it will be satisfied over the optimization period; however, the converse may not hold.

Finally, in varying the profit function, Θ, we cannot define it for each interval since Θ is cumulative over the resource cost, and the VM provisioning may span multiple prediction intervals. Rather, we optimize the profit calculated "so far", given as the cumulative average over each of the previous intervals, i.e., $\Theta^p_{\leq j} = \frac{\forall_{i \leq j} \sum \Gamma^p_i}{\sum P_i} - \mu_{\leq j}$, where $\mu_{\leq j}$ is the cumulative cost till prediction interval P_j. Further, while conciseness precludes a formal proof, it can be shown by contradiction that the global profit function Θ defined earlier is equivalent to $\Theta^p_{\leq k}$, where P_k is the last prediction interval in the given optimization period T.

The look-ahead optimization problem described above assumes a perfect "oracle" prediction model that can see accurately into the entire prediction interval. The predictions obtained at the start of each timestep t_i within the interval are used to *plan* adaptation actions (such as scale up/down instances) for each timestep in the rest of the interval. But these actions are *enacted* only when the timestep is reached.

With a perfect oracle, the predicted values for an interval remain fixed for that interval. However, given the limitations of state-of-the-art multivariate time series models [21], in practice, the errors in the predicted values will increase with the horizon of prediction. Hence, we will get better prediction values as we move further into a prediction interval. To utilize this, we allow new predictions to be made at each timestep t_i rather than at the end of each interval. Further, anytime a new (improved) prediction changes ex-

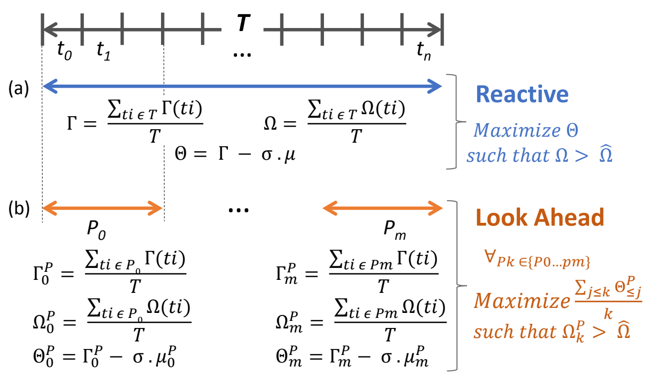

T
t_0 t_1 … t_n
(a)
$\Gamma = \frac{\sum_{ti \in T} \Gamma(ti)}{T}$
$\Omega = \frac{\sum_{ti \in T} \Omega(ti)}{T}$
$\Theta = \Gamma - \sigma.\mu$
Reactive
Maximize Θ
such that $\Omega > \widehat{\Omega}$
(b)
P_0 … P_m
$\Gamma_0^P = \frac{\sum_{ti \in P_0} \Gamma(ti)}{T}$
$\Omega_0^P = \frac{\sum_{ti \in P_0} \Omega(ti)}{T}$
$\Theta_0^P = \Gamma_0^P - \sigma.\mu_0^P$
$\Gamma_m^P = \frac{\sum_{ti \in Pm} \Gamma(ti)}{T}$
$\Omega_m^P = \frac{\sum_{ti \in Pm} \Omega(ti)}{T}$
$\Theta_m^P = \Gamma_m^P - \sigma.\mu_m^P$
Look Ahead
$\forall_{Pk \in \{P0...pm\}}$
Maximize $\frac{\sum_{j \le k} \Theta_{\le j}^P}{k}$
such that $\Omega_k^P > \widehat{\Omega}$

Figure 3. The (a) reactive and (b) look-ahead runtime optimization problems defined over the optimization period T. Reactive optimizes over globally T while look-ahead does it piecewise over each interval P_j. They observe/predict during timesteps t_i and take decisions at the start of t_{i+1}.

isting predicted values, we start a new interval and replan actions for all its timesteps, replacing the prior plan.

This sliding window of replanning does not change the optimization problem, except that the optimization decisions planed in one interval may be altered in the succeeding interval if updated predictions are available. However, note that this is different from the reversal of decisions in the reactive version, since the decisions there are enacted immediately for the next timestep whereas in the look-ahead version the replanning may just change a future action and hence not incur resource penalty.

V. Scheduling Heuristics

As with reactive scheduling (§ III-B), the look-ahead scheduling consists of two phases, the *initial deployment phase* which maps the application to VMs in the Cloud assuming rated performance and that the estimated incoming data rate holds; and the *runtime adaptation phase* which alters the resource allocation and alternates to account for the observed and predicted behavior. In this paper, we retain our earlier initial deployment algorithm [8] and focus on the latter phase. As such, the impact of the initial deployment is amortized by the runtime adaptation for long running dataflows. We first propose a look-ahead scheduling algorithm, *PLAStiCC*, which uses the predicted resource performance and data rates to generate dataflow adaptation plan for the near future. In addition, we propose several simpler heuristics that use averaging models to estimate performance and data rates. These, while out-performing reactive scheduling in many cases (§VI), offer a different trade-off between profit and scheduling cost than PLAStiCC.

A. Predictive Look-Ahead Scheduling (PLASticCC)

The PLAStiCC algorithm is run at the start of each prediction interval. It plans a set of adaptation actions for timesteps in that interval to maximize the cumulative profit while meeting the constraint for that interval. The key intuition is to identify the farthest timestep in the current interval for which the existing deployment meets the QoS constraint for messages expected in that interval, given the performance and data rate predictions. The *cross-over* timestep indicates the future timestep beyond which the current deployment fails to meet the QoS. Once identified, we replan the deployment starting at the cross-over timestep until the last timestep of the interval, recursively. The result of this algorithm is zero or more cross-over timesteps, identified in the interval, where the QoS constraint fails and the planned adaption action to be taken at each timestep.

Algorithm 1 shows the PLAStiCC runtime adaptation algorithm. It takes as input the continuous dataflow DAG D, the predicted data rate $PR^i_{t..t+n}$ for each input PE P_i as a time series $[t..t+n]$ for timesteps in the interval, and the predicted VM performance time series $PV^j_{t..t+n}$ for each VM_i, which includes CPU and inter-VM network behavior. The algorithm outputs a plan of actions X to be taken at timesteps in the prediction interval $[t, t+n]$.

PLAStiCC is called as and when new predictions are made. It is run before the first timestep, t, of the prediction interval and tries to identify the farthest *non*-cross-over timestep. The algorithm maintains two pointers, *ot*, which is the last-known cross-over timestep in the interval, initially blank, and ft, the last timestep of the interval, set to $ft = t + n$. It tries to locate the last timestep when the aggregate constraint is met (lines 5–13). Starting at the last timestep, it calculates the aggregated throughput (Ω) by computing the processing capacity available to each PE (*Alloc*), the cumulative incoming data messages at each PE using the predicted data rates and the network behavior (*M*), and the aggregated relative throughput resulting from these values. If this aggregate throughput is not within $\widehat{\Omega} \pm \epsilon$, we try the previous timestep (line 12). If it is met, have found the cross-over boundary between constraint being satisfied and not (line 10). If the constraint is met in the first iteration, i.e. at the interval's end, no adaptation is needed (line 15).

Once the cross-over is identified, *ot* is set to this last-known cross-over timestep, and ft reset to the interval end $t+n$ for the next iteration. We then decide the action to be taken at the cross-over to meet the constraint. For this replanning, we get the aggregated values for the available resources, the incoming messages and the relative throughput for the rest of the interval after the cross-over, $(ot, t+n]$ (lines 18–20). Since the expected throughput is deviating for this period, we either *scale up* ($\Omega_{ot..t+n} < \widehat{\Omega} - \epsilon$) or *scale down* ($\Omega_{ot..t+n} > \widehat{\Omega} + \epsilon$)), *incrementally* (lines 21–27).

Both the ScaleUp and the ScaleDown functions try to balance the overall application value Γ (Sec. III-A) and the total application cost μ. The ScaleUp function first identifies the "bottleneck" PE with minimum relative throughput, lying on the critical path from the input to output PEs [8]. It can then either switch to an alternate for this PE with lower processing needs (and potentially decrease the application value) or increase the number of instances (and possibly increase the resource cost) for the bottleneck PE. This choice is based on the local optimization of the profit function Θ_t which uses the user specified σ coefficient that determines the "acceptable" value:cost ratio.

Specifically, the cost of increasing an instance is zero if a VM has spare cores or else is equal to the cost of a new VM. Then, the change in application value from switching to the alternate with next lower processing needs is calculated. The change in Θ_{ot-ft} from either of these actions is calculated, and the adaptation that results in higher profit is selected. Similarly, the SCALEDOWN function identifies an over-provisioned PE and makes a choice between switching to an alternate with higher value or decreasing the number of instances, whichever leads to higher profit.

Once such incremental action is planned for the cross-over timestep, we resume evaluating the planned deployment ensure it satisfies the constraint for the rest of the interval, $(ot, t{+}n]$ (lines 5 - 13). The replanning is incrementally done till all cross-over timesteps and adaptations are identified for the interval, and the resulting plan satisfies the constraint for the entire interval. Finally, the algorithm performs a repacking of VMs to collocate distributed PEs as well as shuts down those that are unused (line 30).

Algorithm 1 PLAStiCC Runtime Adaptation Heuristic

```
1: procedure PLASTICC(Dataflow $D$, MinThroughput $\widehat{\Omega_t}$, DataRatePredictions $PR_{t..t+n}$, VMPerfPredictions $PV_{t..t+n}$)
2:     ot ← ∅, X ← ∅                               ▷ Init cross-over list to null
3:     while ot ≤ t+n do
4:         ft ← t+n
5:         while ft ≥ t do                          ▷ Work back from last timestep.
6:             Alloc ← AVAILABLERESOURCEALLOC($PV_{ot..ft}$)
7:             M ← CUMULATIVEINCOMINGMSGS($PR_{ot..ft}$)
8:             Ω ← CUMULATIVETHROUGHPUT(D, M, Alloc)
9:             if $\widehat{\Omega} - \epsilon < \Omega < \widehat{\Omega} + \epsilon$ then
10:                Break                             ▷ QoS constraint met at this timestep.
11:            end if
12:            ft ← ft - 1                           ▷ QoS not met. Work backwards.
13:        end while
14:        if ft = t+n then                          ▷ QoS met at end of interval.
15:            Break                                 ▷ No more replanning. Done.
16:        end if
         ▷ Replan actions for the remaining duration $(ft, t+n]$
17:        ot ← ft                                   ▷ Update last-known cross-over.
18:        Alloc ← AVAILABLERESOURCEALLOC($PV_{ot..t+n}$)
19:        M ← CUMULATIVEINCOMINGMSGS($PR_{ot..t+n}$)
20:        Ω ← CUMULATIVETHROUGHPUT(D, M, Alloc)
21:        if $\Omega < \widehat{\Omega}$ then
22:            bpe ← BOTTLENECK(D,CA)
23:            X ← SCALEUPORDECAPPVALUE(bpe, ot)
24:        else if $\Omega > \widehat{\Omega}$ then
25:            ope ← OVERPROVISIONED(D,CA)
26:            X ← SCALEDOWNORINCAPPVALUE(bpe, ot)
27:        end if
28:    end while
29:    X ← REPACKANDSHUTDOWNVMS( )
30:    return X                     ▷ Return cross-over timesteps and action plan.
31: end procedure
```

B. *Averaging Models with Reactive Scheduling*

The PLAStiCC algorithm reduces to the reactive algorithm if the length of the prediction interval is exactly one timestep, i.e. only current and no future values for data rates and infrastructure performance are given. Rather than use fine-grained predictions (PLAStiCC) or assume that the most recent observed values will sustain (reactive), we develop several heuristics that use simple but intuitive estimates of future behavior based on aggregated past behavior. This also addresses PLAStiCC's susceptibility to prediction errors since it overfits the schedule to the predictions. Similar to the reactive scheduler, these heuristics are executed only when the application QoS deviates from the constraint. The heuristics each differ in the type of averaging done over observed values to estimate future behavior of resource performance and data rates.

1) *Current After-the-fact (CAF)* This chooses observed value in the most recent timestep as the estimated future behavior for all timesteps. This is same as the classic reactive scheduler [8]. This reacts after-the-fact to the observed resource and data rate changes.
2) *Windowed Average After-the-fact (WAAF)* This uses the average seen over several recent timesteps as the future estimated behavior. This smooths out spikes while accounting for the recently observed trends that might last in the near future.
3) *Windowed Predictive Average (WPA)* This averages the values predicted for all timesteps by the advanced prediction model used by PLAStiCC, and uses the single average value in all timesteps in the interval. This mitigates the impact of large prediction errors in a few timesteps, where the PLAStiCC is susceptible.
4) *Windowed Predictive Optimistic (WPO)* This too uses the advanced prediction model of PLAStiCC but picks the most favorable infrastructure performance (highest) and data rate (lowest) from among all timesteps predicted for, and uses that single value for all timesteps. This offsets the impact of models that under-predict [22].
5) *Windowed Predictive Pessimistic (WPP)* This is similar to WPO, except it picks the least favorable performance and data rate prediction as the single value. This handles models with over-prediction bias [22].

VI. EVALUATION

Simulation Setup: We evaluate PLAStiCC and the different averaging models for reactive scheduling through a simulations study. We extend the CloudSim [23] data center simulator to *IaaSSim* [2], that additionally incorporates temporal and spatial performance variability using real traces

[2]http://github.com/usc-cloud/IaasSimulator

collected from IaaS Cloud VMs. Further, our *FloeSim* [3] simulates the execution of our Floe stream processing engine [20], on top of IaaSSim, with support for continuous dataflows, alternates, distributed deployment on VMs, dynamic instance scaling and a plugin for different schedulers.

We use Amazon EC2's performance traces (§ II) in IaaSSim. For each simulation experiment, we deploy a dataflow to the Cloud using FloeSim, run it for 12 hours (simulated) at a stretch – which is also the optimization period ($T = 12hrs$), and repeat it several times to get an average. We use a timestep duration of $t = 5mins$. Experiments vary the scheduler used, and, for PLAStiCC, the prediction interval changes between $P = 10 - 30mins$.

Synthetic Dataflows and Streams: We evaluate the adaptation algorithms using a synthetic dataflow in Figure 4 with 10 stages of 3 PEs each, and 3 input and output data streams. The dataflow is randomly perturbed to generate between 1–3 alternates each, with different cost, value and selectivity. This gives us a total of 30 PEs and ∼45 alternates.

Each of the three input data streams are independently generated based on observations in domains like Smart Power Grids [18]. Starting from a base stream with sinusoidal data rates whose peak and trough range from $75 - 2$ messages, and with wave length of $24hrs$, we add random noise every $1min$ that varies the rate by up to $\pm 20\%$.

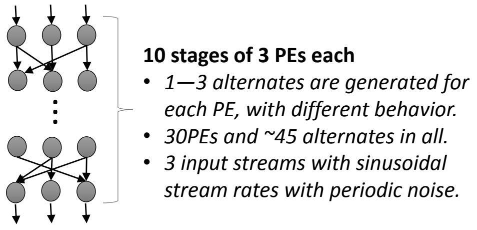

10 stages of 3 PEs each
• 1—3 alternates are generated for each PE, with different behavior.
• 30PEs and ~45 alternates in all.
• 3 input streams with sinusoidal stream rates with periodic noise.

Figure 4. Sample dataflow pattern used for evaluation

Evaluation Metrics

Overall Application Relative Throughput (Ω):: This is the aggregate relative throughput obtained at the end of the optimization period. We check if the heuristic satisfies the QoS constraint $\Omega \geq \widehat{\Omega}$. A higher value of Ω beyond $\widehat{\Omega}$ is not necessarily good unless it also gives a higher overall profit.

Overall Application Profit (Θ): : This is the aggregate profit $\Theta = \Gamma - \sigma \times \mu$ obtained at the end of the optimization period. Given two algorithms which satisfy the QoS constraint, the one with a higher overall profit is better. Note that profit is *unitless* and can only be compared relatively.

Heuristic Stability (χ):: This is the frequency of deviation in instantaneous relative throughput Ω_t from the $\widehat{\Omega} \pm \epsilon$ during the optimization period. For reactive algorithms, this is the number of times adaptation actions respond to the deviation, while for PLAStiCC, this is the number of time the *planned* actions failed to meet the constraint. A lesser number indicates a more stable heuristic.

[3]http://github.com/usc-cloud/FloeSim

A. Experiments

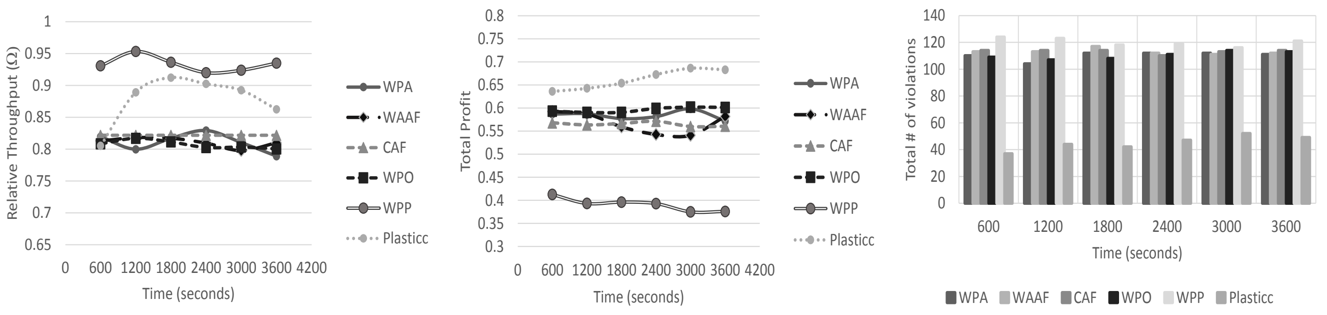
1) Scheduling under Ideal Predictions: We evaluate the proposed heuristics for a perfect oracle model where the predictions for resource performance and stream rates are accurate. Further, we evaluate them with prediction intervals ranging from 10 mins (i.e. two $5min$ timesteps per interval) upto to 30 mins and study the effect of the length of prediction interval under perfect oracle.

2) Scheduling under Predictions with Bounded Errors: We run experiments where the model predicted performance and data rate values include bounded errors, $\hat{e}$, to simulate short-comings of real world prediction models. The predictions have smaller error for near timesteps and larger ones for farther timesteps. First, we linearly scale the error from *zero* for the current timestep (e_0) to the $2 \times \hat{e}$ for the last timestep (e_n) in the prediction interval, to give a mean close to $\hat{e}$. Then, we sample the actual error for a timestep i from a normal distribution with mean 0 and variance equal to $e_i/2$, and use this as the actual error for that timestep to obtain the predicted value. We can also optionally specify a "bias" of $\pm 5\%$ that is added to get a positive or negative biased model. We make predictions every $10mins$ for all models.

B. Results

Perfect Oracle Model: We first present the results obtained for the perfect oracle model, which signifies the best case results for the PLAStiCC heuristic as compared to the various reactive models. Figure 5 shows comparison between the PLAStiCC algorithm to the reactive models for different metrics (Ω, Θ, and χ) with different prediction interval lengths for which the oracle predictions are available). Figure 5(a) shows the overall relative throughput for various algorithms observed at the end of the optimization duration. We observe that all the adaptation algorithms successfully satisfy the application constraint ($\Omega \geq 0.8$) over the optimization duration for prediction interval lengths.

Higher values of relative throughput do not necessarily mean a better algorithm. We compare this with Θ, and χ. Figure 5(b) shows the overall profit ($\Theta = \Gamma - \sigma\mu$) obtained for different algorithms. First we observe that the PLAStiCC algorithm performs consistently better than any of reactive algorithms across different prediction interval lengths. Further, we also observe that the profit value for the PLAStiCC algorithm increases as the length of the prediction interval increases. This is because, with shorter prediction intervals, the planned actions for that interval may be already executed before the plan for the future is obtained, and hence leads to a reversal in the actual action and not just the planned action, the former of which incurs additional cost. As the length of the interval increases, fluctuations further in the future are known and hence the planned activities may be reversed before they are enacted and hence lowering the resource cost. In addition, we observe that the heuristics based on current (CAF) and historical average (WAAF) performance



(a) Overall Relative Throughput (Ω) (b) Total Profit (Γ) (c) Heuristic Stability (χ)

Figure 5. Performance of scheduling for a perfect, oracle prediction model, as the prediction interval horizon (in secs) vary on X axis

perform fairly similar. However, the WPP algorithm leads to much lower profit since it always tries to allocate resources for the worst case scenario, hence incurring higher resource cost. Further, Figure 5(c), shows the stability of the different heuristics (lower values are better). We observe that the PLAStiCC algorithm is more stable than any of the other heuristics and hence provides more predictable performance over the optimization duration. This is mainly because the algorithm pro-actively takes actions (such as launch VMs in advance to account for VM launch time while initiating a new instance at the desired time), whereas in the reactive versions the delay incurred by such actions cause constraint mismatch till the action is performed.

This simulation study thus show that the look-ahead PLAStiCC algorithm out-performs all the reactive algorithms in the presence of data and infrastructure variability. Further, it shows that the PLAStiCC algorithm is dependent on the length of the prediction interval and performs better as the interval length increases.

Predictions with Errors and Biases: Next, we analyze PLAStiCC under a more realistic prediction model conditions, with prediction errors. We compare PLAStiCC against WPO, WPA, and WPP to evaluate their relative strengths.

Figure 6 shows the performance of PLAStiCC for models with prediction errors but no biasing. As before, we observe that the profit of the algorithm for a perfect oracle model (0% error) increases from 0.6 to 0.7 with increase in the prediction interval from $10 - 60 mins$. However, we see degrading performance as the error rate increases to 20%. We make two observations. First, for a given prediction interval length the performance decreases, as expected, as prediction error increases. Second, with high error values ($> 10\%$), the performance of the algorithm, for a given error rate, initially increases with the prediction interval length, and then falls with further increase in the interval length (inverse bell curve). The reason for such behaviour is that as we predict farther into the future, the error increases. But as we move into the future, more accurate results are obtained and hence they lead us to reverse actions. While most of these are planned actions, some enacted ones may be reversed too. As the error increases, only prediction very

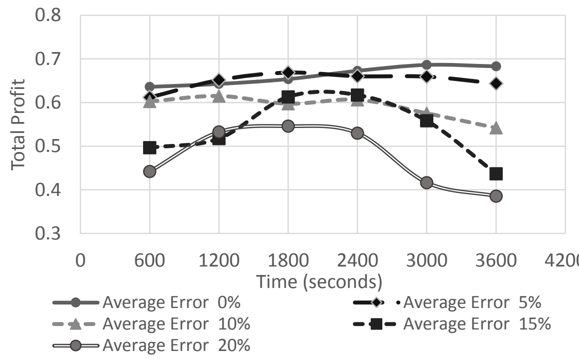
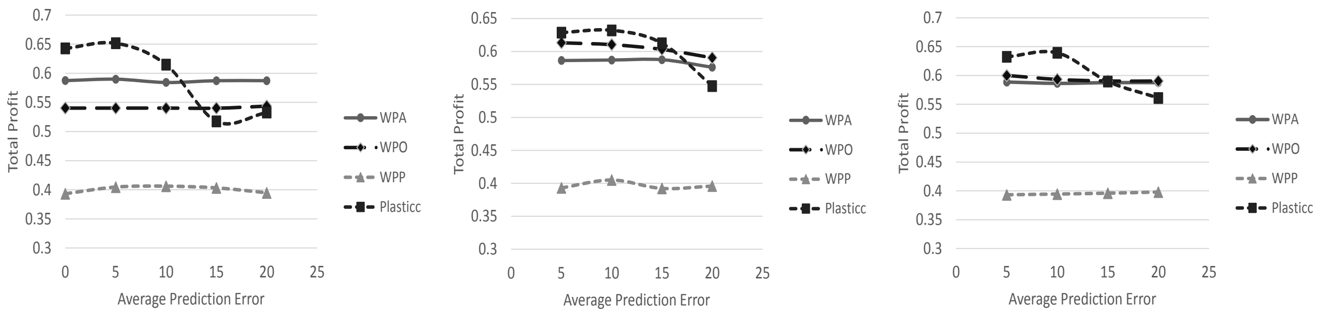


Figure 6. Overall profit of PLAStiCC for different prediction error %, with prediction interval horizon (in secs) on X axis

close to the current time will prove accurate and hence the PLAStiCC degenerates to the reactive version.

Finally, we compare PLAStiCC against the reactive versions (WPA, WPO & WPP) which aggregate over values predicted by the model (rather than observed), and are potentially affected by the prediction error. CAF and WAAF are unaffected by prediction errors and hence skipped. Figure 7 shows the overall profit for WPA, WPO, WPP and PLAStiCC for different error rates (X axis) and prediction biases (a-c). A prediction interval of length 20 mins is chosen for these experiments. As seen in the figures, PLAStiCC outperform the others for smaller errors, irrespective of the prediction bias, while its performance degrades as the error increases, since it tries to overfit to the predicted values. WPA, on the other hand performs consistently with different error rates and for different biases. While the WPO algorithm, which assumes an optimistic infrastructure behaviour, performs poorly when the prediction errors are unbiased. However, it performs better than WPA if the predicted values are lower than the actual values. The reason is apparent. With lower predicted values, the best performance values over the interval tend to be closer to the real performance value than the average. On the other hand, WPP performs poorly under all scenarios. This is because the WPP assumes pessimistic infrastructure performance. A single dip in the predicted performance causes it to over-provision resources.



(a) Overall profit for different algorithms for *unbiased* predictor (0%) (b) Overall profit for different algorithms for *low biased* predictor (-5%) (c) Overall profit for different algorithms for *high biased* predictor (+5%)

Figure 7. Performance of scheduling with realistic prediction models, having different prediction error % on X axis. Plots differ in prediction bias.

VII. Related Work

A. Continuous Dataflows

Continuous dataflows have emerged as an extension to the Data Stream Managements systems, which were focused on continuous queries over the data streams. These have evolved into a general purpose abstraction that provides strong composition tools for building large scale stream processing applications. Several such stream processing systems have been recently proposed such as S4 [4], IBM InfoSphere streams [17], D-Streams [5], Storm [24], and Time Stream [6]. While these systems are built for large scale stream processing applications and provide distributed deployment and runtime, most of these (with the exception of Time Stream) do not provide automatic, dynamic adaptations to changing data rates or performance variability of the underlying infrastructure, over the application's lifetime.

B. Dynamic Adaptations and Scaling

The notion of "alternates" is similar to flexible workflows [25]. It also draws inspiration from dynamic tasks defined in heterogeneous computing [19] where a task is composed of one or more alternatives with different characteristics, such as resource requirements and dependencies. However, unlike these systems, where the alternate to be executed is selected in advance, in our dynamic continuous dataflow abstraction such decisions are made at periodic intervals based on the changing execution characteristics. Time Stream [6] supports similar dynamic reconfiguration, called *resilient substitution*, which allows replacing a subgraph with another in response to the changing load. This requires dependency tracking and introduces inefficiencies. In contrast, we restrict dynamic adaptation to single processing elements making the process of dynamic reconfiguration independent for each PE and hence allow more flexibility in dynamic adaptation, while restricting the application composition model. Our earlier work has discussed consistency models for enacting such runtime updates [26].

C. Predictive Scheduling

The predictive scheduling approach proposed in the paper follow the generic "scheduler, monitor, comparator, resolver" approach proposed by Nof et. al. [27]. Several such predictive scheduling techniques have been studied on computation grids [28], [12], however in the context of batch task scheduling instead of continuous dataflows. Further, several performance prediction models for the Grid have been proposed to profile workloads as well as infrastructure variations [29]. Although, we do not provide models for performance prediction in the Clouds, we envision that similar performance models may be applicable. PRESS [14] is one such system which uses "signature driven" prediction for variations, with periodic patterns and a discrete-time Markov chain based model for short-term predictions for VMs. They achieve high prediction accuracy with less than 5% over-estimation error and close to zero under-prediction error. We presume existence of such monitoring and prediction algorithm to be used as the basis for our look-ahead scheduling algorithm.

VIII. Conclusion

In this paper, we analyzed the temporal and spatial performance variations observed in public and private IaaS Clouds. We proposed PLAStiCC, a predictive look-ahead scheduling heuristic for dynamic adaptation of continuous dataflows, that responds to fluctuations in stream data rates as well as variations in Cloud VM performance. Through simulations, we showed that the proposed PLAStiCC heuristic results in a higher overall throughput, up to 20%, as compared to the reactive version of the scheduling heuristic, we proposed earlier and improved here, which performs adaptation actions after observing the effects of these variations on the application. We also studied the effect of realistic prediction models with different error bounds and biases on the proposed heuristic, and identified scenarios where the look-ahead algorithm falls short of the reactive one.

References

[1] W. Yin, Y. Simmhan, and V. Prasanna, "Scalable regression tree learning on hadoop using openplanet," in *MAPREDUCE*, 2012.

[2] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *SIGMOD*, 2010.

[3] B. Satzger, W. Hummer, P. Leitner, and S. Dustdar, "Esc: Towards an elastic stream computing platform for the cloud," in *CLOUD*, 2011.

[4] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *ICDMW*, 2010.

[5] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters," in *HotCloud*, 2012.

[6] Z. Qian, Y. He, C. Su, Z. Wu, H. Zhu, T. Zhang, L. Zhou, Y. Yu, and Z. Zhang, "Timestream: Reliable stream computation in the cloud," in *ECCS*, 2013.

[7] D. Zinn, Q. Hart, T. M. McPhillips, B. Ludscher, Y. Simmhan, M. Giakkoupis, and V. K. Prasanna, "Towards reliable, performant workflows for streaming-applications on cloud platforms," in *CCGRID*, 2011.

[8] A. Kumbhare, Y. Simmhan, and V. Prasanna, "Exploiting application dynamism and cloud elasticity for continusous dataflows," in *SC*, 2013.

[9] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *CCGrid*, 2011.

[10] H.-Y. Chu and Y. Simmhan, "Resource allocation strategies on hybrid cloud for resilient jobs," USC, Tech. Rep., 2013.

[11] B. Lu, A. Apon, L. Dowdy, F. Robinson, D. Hoffman, and D. Brewer, "A case study on grid performance modeling," in *ICPDCS*, 2006.

[12] R. M. Badia, F. Escale, E. Gabriel, J. Gimenez, R. Keller, J. Labarta, and M. S. Müller, "Performance prediction in a grid environment," in *Grid Computing, LNCS*, 2004.

[13] D. Spooner, S. Jarvis, J. Cao, S. Saini, and G. Nudd, "Local grid scheduling techniques using performance prediction," *Computers and Digital Techniques*, vol. 150, no. 2, 2003.

[14] Z. Gong, X. Gu, and J. Wilkes, "Press: Predictive elastic resource scaling for cloud systems," in *CNSM*, 2010.

[15] R. P. Weicker, "An overview of common benchmarks," *IEEE Computer*, vol. 23, no. 12, 1990.

[16] R. Wolski, N. T. Spring, and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," *FGCS*, vol. 15, no. 5, 1999.

[17] A. Biem, E. Bouillet, H. Feng, A. Ranganathan, A. Riabov, O. Verscheure, H. Koutsopoulos, and C. Moran, "Ibm infosphere streams for scalable, real-time, intelligent transportation services," in *SIGMOD*, 2010.

[18] Y. Simmhan, S. Aman, A. Kumbhare, R. Liu, S. Stevens, Q. Zhou, and V. Prasanna, "Cloud-based software platform for data-driven smart grid management," *CiSE*, 2013.

[19] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *JPDC*, vol. 59, no. 2, 1999.

[20] Y. Simmhan, A. Kumbhare, and C. Wickramachari, "Floe: A dynamic, continuous dataflow framework for elastic clouds," USC, Tech. Rep., 2013.

[21] K. Chakraborty, K. Mehrotra, C. K. Mohan, and S. Ranka, "Forecasting the behavior of multivariate time series using neural networks," *Neural networks*, vol. 5, no. 6, 1992.

[22] P. Krause, D. Boyle, and F. Bäse, "Comparison of different efficiency criteria for hydrological model assessment," *Advances in Geosciences*, vol. 5, no. 5, 2005.

[23] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, 2011.

[24] S. Loesing, M. Hentschel, T. Kraska, and D. Kossmann, "Stormy: an elastic and highly available streaming service in the cloud," in *EDBT/ICDT Workshops*, 2012.

[25] A. H. H. Ngu, S. Bowers, N. Haasch, T. M. Mcphillips, and T. Critchlow, "Flexible Scientific Workflow Modeling Using Frames, Templates, and Dynamic Embedding," in *SSDBM*, 2008.

[26] C. Wickramachari and Y. Simmhan, "Continuous dataflow update strategies for mission-critical applications," in *eScience*, 2013.

[27] S. Y. NOF and F. HANK GRANT, "Adaptive/predictive scheduling: review and a general framework," *Production Planning & Control*, vol. 2, no. 4, 1991.

[28] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox, "Pace–a toolset for the performance prediction of parallel and distributed systems," *Int. J. HPCA*, vol. 14, no. 3, 2000.

[29] Y. Wu, K. Hwang, Y. Yuan, and W. Zheng, "Adaptive workload prediction of grid performance in confidence windows," *TPDS*, vol. 21, no. 7, 2010.