Gautam and Basava
Hum. Cent. Comput. Inf. Sci. (2019) 9:2
https://doi.org/10.1186/s13673-018-0163-4

Human-centric Computing and Information Sciences

RESEARCH Open Access

Performance prediction of data streams on high-performance architecture

Bhaskar Gautam* and Annappa Basava

*Correspondence: bhaskar.gautam2494@gmail.com
Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal, India

Abstract

Worldwide sensor streams are expanding continuously with unbounded velocity in volume, and for this acceleration, there is an adaptation of large stream data processing system from the homogeneous to rack-scale architecture which makes serious concern in the domain of workload optimization, scheduling, and resource management algorithms. Our proposed framework is based on providing architecture independent performance prediction model to enable resource adaptive distributed stream data processing platform. It is comprised of seven pre-defined domain for dynamic data stream metrics including a self-driven model which tries to fit these metrics using ridge regularization regression algorithm. Another significant contribution lies in fully-automated performance prediction model inherited from the state-of-the-art distributed data management system for distributed stream processing systems using Gaussian processes regression that cluster metrics with the help of dimensionality reduction algorithm. We implemented its base on Apache Heron and evaluated with proposed Benchmark Suite comprising of five domain-specific topologies. To assess the proposed methodologies, we forcefully ingest tuple skewness among the benchmarking topologies to set up the ground truth for predictions and found that accuracy of predicting the performance of data streams increased up to 80.62% from 66.36% along with the reduction of error from 37.14 to 16.06%.

Keywords: Apache Heron, Stream benchmark suite, Performance prediction, Performance behavior, High performance computing, Regression, Clustering, Data streams

Introduction

The specialized distributed real-time stream processing systems demand the underlying system should able to adapt with increment in data volume, having heterogeneous data sources. Along with the requirement of massive computational capabilities on increasing data velocity, these specialized systems also insist that underlying framework should provide highly scalable resources to achieve massive parallelism among the processing logic components in a distributed computing nodes in a timely manner and to facilitate fast recovery from hardware failures, stateless and stateful mechanism of processing logic components ensure low latency streaming. Among all state-of-the-art specialized distributed stream processing framework Apache Storm [1], Apache Flink [2], and Apache Spark have emerged as the de facto programming model which automatically take care of data and process distribution to achieve sufficient task parallelism. More recent forays in low-latency distributed stream processing, Google MillWheel [3] and Apache Heron [4] emerged as a successor to all modern unbounded streams of continuous data processing systems and scale transparently to large clusters which are most common among all stream processing engines. Although there are similarities among components but they provide a different mechanism such as tuples or buffers for message passing to provide high throughput.

The emerging real-time distributed stream processing system's *Heron* is built from plethora of components named as *Spout*, *Bolts*, *Topology Master*, *Stream Manager* and *Metrics Manager* which interacts in complex ways while running on several containers to correlate with high velocity of data volume. These containers are scheduled to run on a heterogeneous selection of multi-core nodes using large-scale storage infrastructures. It also provides a framework to seamlessly integrate with existing large data processing components named as Apache Hadoop Distributed File System, Apache REEF [5], Apache Mesos, Apache Aurora [6], Simple Linux Utility for Resource Management (SLURM) and Google Kubernetes [7] but simultaneously makes it difficult to understand the performance behavior of underlying applications and components. Traditional relational database management systems performance complexities can be resolved using optimizers [8] but how to accurately model and predict performance complexities in distributed stream processing framework is quite challenging and has not yet been well studied. We address this gap in this paper. These performance complexities arise due to huge variance in workloads, elasticity, computation fluctuations and tuple serialization rate which makes difficult to predict the behavior of data pipelined on distributed components. Since predicting the dynamic performance of data stream will provide further insight to a number of data management task including workload optimization [9], scheduling [10] and resource management which help in reducing unnecessary over-provisioning of resources through efficient prioritization of resource allocations in the specialized distributed stream processing systems domain.

In this paper, we propose a novel architecture independent performance prediction framework for text streaming in distributed stream processing platform running on top of OpenHPC systems. Specifically, we summarize our contribution to the following:

- We provide domain specific metrics which were most relevant for streaming platform running on top of high performance computing architecture because existing methodologies only depicts about the big data processing and distributed database management framework.
- We provide performance behavior of streaming platform running on top of high performance architecture.
- We transform state-of-the-art automated performance tuning module of distributed database management system to work for distributed streaming platform.

- We propose a novel framework running on top of a streaming platform using linear least squares with L2 regularization to recommend a plausible performance for the stream of individual topology.
- To validate and evaluate the proposed framework, we implemented on an emerging processing system, Apache Heron.

The remainder of this paper is structured as follows: in "Background" section, we present the adequate background for the entire paper. "Design and implementation of proposed framework" section presents the methodology followed by an overview of the proposed framework. "Experimental evaluation" section presents the evaluation and results while comparing with all proposed models. In "Related work" section, all related literature are discussed and finally, the paper concludes in "Discussions and conclusions" section with addressing some of the conceivable used cases.

Background

A standard stream processing framework running on high performance computing cluster is the one where every component is running on computing nodes, an exemplary architecture described in Fig. 1. The processing representation of the continuous progression of tuples, streams we model it into a directed acyclic graph (DAG). These acyclic graphs are well known as topology in heron and have the capacity for processing of these tuples with ∞ number of times which eventually depends on the availability of tuples. Topologies in heron comprise three basic processing logic components (PLU) labeled as *spout*, *bolt* and *edging bolt*. The source processing component, spout read tuples potentially from the outsourced stream publisher-subscriber system (here, *Apache Pulsar* [11]) and seeds tuples into a contemporary graph having count zeros in-degree. The tuple processing component, bolt parse the seeded tuple with user-defined processing logic and later, seed the processed tuple into a contemporary graph such that in-degree ≥ 1 and out-degree ≥ 1 to maintain the stream processing pipeline. Similarly, edging bolt or sink processing component parses the seeded tuple with user-defined processing logic and later, seed the processed tuple into outsourced storage such that in-degree ≥ 1 and $out\text{-}degree \Leftarrow 0$ to maintain the stream processing pipeline. The vertices in the logical plan of topology represent nodes of contemporary graph and direction of these vertices represents the progression of these tuples whole scenario is elaborated in Fig. 1. These processing logic component instances packed into a containerized process, *Heron Instance* which can able to execute as many parallel tasks on multiple containers hosted on either single or multiple computing nodes. These user-defined topologies are distributed to the cluster through one of the scalable mechanism named as Hadoop File System, Local File System, and Lustre File Systems.[1] Dynamically, the efficiency of contemporary topologies is maintained using the back-pressure mechanism [12] for *spout* and *bolt* respectively maintained through *Topology Master*. Tuples in this framework are generally composed of the message with the encoded meta-attributes object. Heron has

[1] http://www.lustre.org.

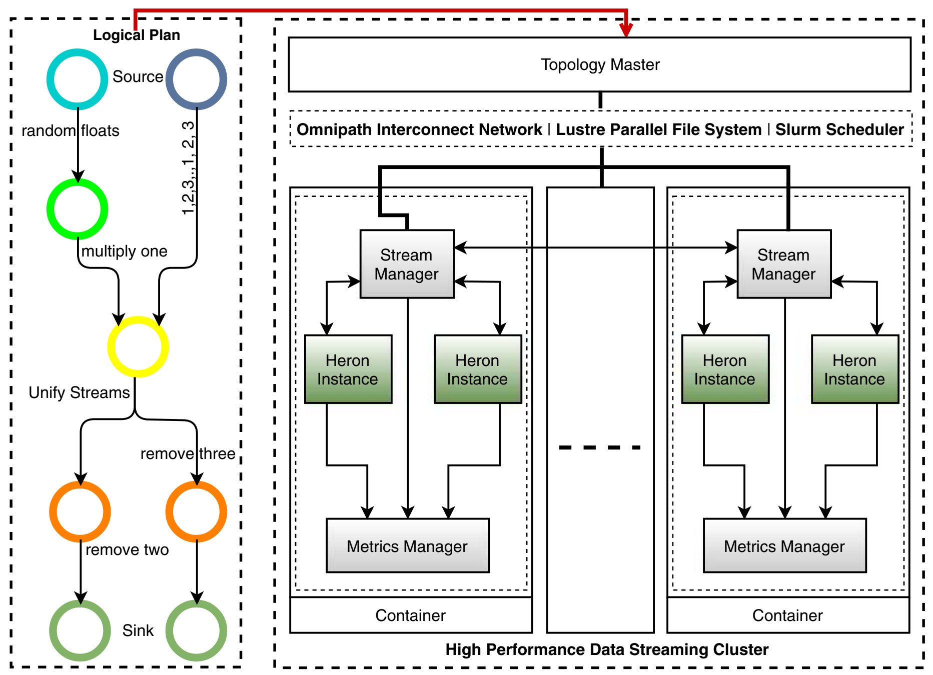

Fig. 1 The exemplary architecture of a high performance data stream processing system

six ways of grouping the tuples among contemporary processing component described as follows:

- Fields grouping: The progression of tuples is transmitted to those processing logic components comprised of similar meta-attribute value.
- Global grouping: The progression of tuples is transmitted to single instance having lowest encoded meta-attribute value.
- Shuffle grouping: The progression of tuples is randomly distributed to distinct instances while ensuring uniform distribution.
- None grouping: Till now, having similar functionality as shuffle grouping.
- All grouping: The progression of tuples distributed to all corresponding processing components.
- Custom grouping: The progression of tuples distributed to corresponding processing components as defined by the user.

Heron has gathered a results in following two ways described as follows:

- Sliding window: Tuples in a stream are grouped together to form windows that can be overlap either on the basis of time duration or on number of operation performed.
- Tumbling window: Tuples in a stream are grouped together to form non-overlapping window either on the basis of time duration or on number of operation performed.

A distributed stream data processing system consist of master node that serves as the topology life cycle management unit and helps in transformation of logical plan into physical plan which are analogous to a database query plan [13] using state-of-the-art bin packing algorithm. Inspired from microkernel based architecture, it share these plans to the *SLURM scheduler* which further assign the task to specific compute nodes as per the physical plan of a topology. The scheduling solution leads to almost even distribution among all containers which assign a task to instances in a round-robin manner.

Design and implementation of proposed framework

In this section, we describe the overview of design and implementation details of the proposed framework.

Overview

Our proposed benchmark suite allows to inject various data loads into heron stream data processing systems and collect dynamic metrics which helps to estimate runtime performance of data streams in streaming topology. The overall architecture is presented in Fig. 2 and comes with modules such as *Metrics Pipeline*, *Data Enrichment*, *Metrics Classification*, *Data Grid*, *Data Stream Performance Prediction Model* and *Benchmark Suite*. It consists of five stream processing benchmark topologies to cover wide domains with help of CPU Intensive topologies, Memory Intensive topologies, Network Intensive topologies and Scheduler Intensive topologies. Conceptually, each benchmarking

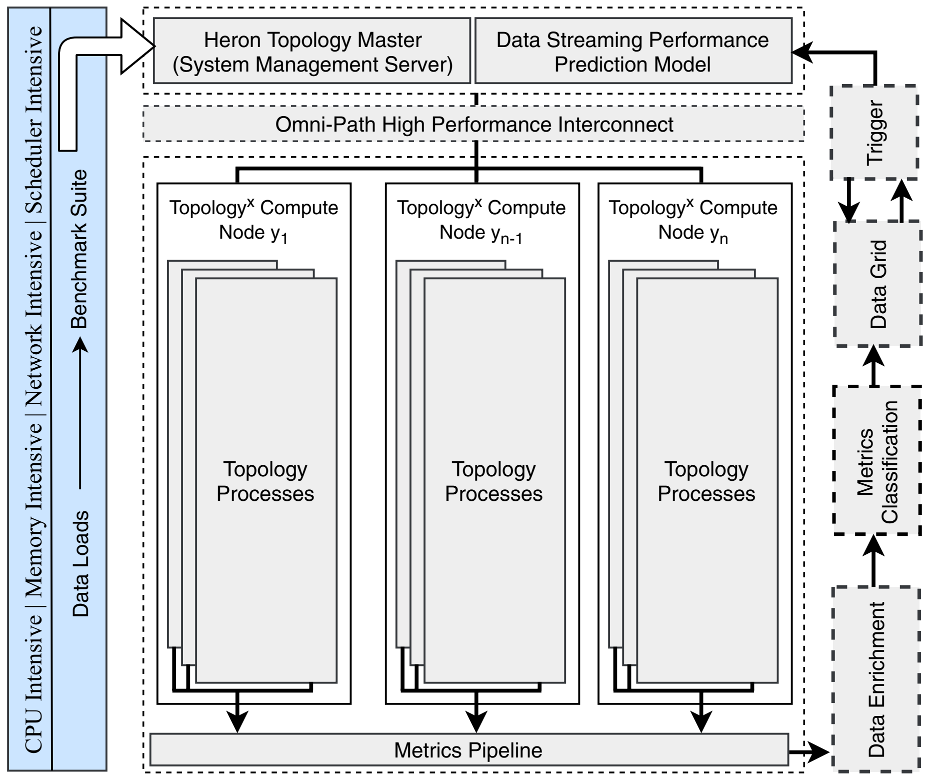
CPU Intensive | Memory Intensive | Network Intensive | Scheduler Intensive
Data Loads
Benchmark Suite
Heron Topology Master (System Management Server)
Data Streaming Performance Prediction Model
Omni-Path High Performance Interconnect
Topology^x Compute Node y_1
Topology^x Compute Node y_{n-1}
Topology^x Compute Node y_n
Topology Processes
Topology Processes
Topology Processes
Metrics Pipeline
Trigger
Data Grid
Metrics Classification
Data Enrichment

Fig. 2 An overview of the incremental performance prediction framework where lightly shaded region represents the performance advisor model flow

topology implements services running on a high performance stream data processing cluster where each service has several types of requests issued by users.

Performance metrics classification

There is no single metric exists till the time of writing paper based on which we evaluate overall performance of big data system which is almost the same problem discussed in [14]. In this section, we are classifying all existing metrics into seven different categories which helps in deeper visualization of strength and weakness of entire big data processing systems discussed in following section.

Memory metrics

Heap memory Running topology in containerized environment contains another layer of abstract execution environment on top of hardware virtualization over a physical hosting platform and sharing these hardware resources along with memory among multiple Java virtual machines (JVMs) ends up with the unpredictable memory demands as discussed in [15].

$$Heap\,available\,(\%, \mathrm{mb}) = \frac{\mathrm{Heap\ free}}{\mathrm{Heap\ total}} \times 100 \tag{1}$$

To consider such memory behavior, the percentage of Heap Available metric is computed using the total amount of heap memory free divided by total amount of heap memory available in terms of megabytes.

Garbage collection time The total accumulated milliseconds time spent by the garbage collector managed bean (MBean) [16] to find and reclaim unreachable objects to free up memory space per minute known as garbage collection time (GC_{Time}).

$$GC_{Time} = \alpha + \beta + \gamma \tag{2}$$

Alternatively, GC_{Time} defined as the total accumulated time spent to determine the number of reachable objects (α), count of unreachable objects (β) and time to free up memory space in a milliseconds window frame (γ).

n-Verticals metrics

Thread share The total actively running live threads are simultaneously made request for the services in the same container at the given instance of time. These accumulated running threads also comprising of background supporting task which are fulfilled by daemon threads.

$$Threads\,share\% = \frac{|Active\_Threads - Daemons|}{Active\,threads} \times 100 \tag{3}$$

A total number of active threads (non-daemons) count at the given time can be evaluated as modulus subtraction of numbers of active threads with a number of active daemon threads running. Later, fraction with total active thread which is known as thread share.

CPU load The allocated cores which are actively running process (p) for x duration of time from the recent period being observed are termed as CPU Load of containerized process.

$$Core\ idle\% = (1 - Process\_Load) \times 100 \tag{4}$$

Alternatively, process CPU load is defined as a product of current process CPU load with the number of allocated cores to the topology. And the range of such java virtual machine variable count lies between 0 and 1. Hence, the percentage of core idle is related to the processing load and is defined as the complement of process load.

Communication metrics

Back pressure The total accumulated time spent by an instance under back-pressure.

$$BPTime = \theta_1 + \theta_2 + \theta_3 \tag{5}$$

We measure back-pressure (BP) time [17] in terms of milliseconds per minute which includes TCP back-pressure (θ_1), spout back-pressure (θ_2) and stage-by-stage back-pressure (θ_3) as heron internal includes both back-pressure initiated by self and others.

Computation metrics

Execute latency The execution latency is the latency it acquired to process a user-defined logic on windowed incoming tuples of a topology.

Scheduler metrics

Uptime The total computation time allocated to a process on which Java virtual machine is running, once shortlisted by the short-term scheduler. In rest of the paper, we keep nanoseconds as a unit of measurement in metrics pipeline module .

Among all the selected metrics, containerized configuration as a cost metrics (RAM, CPU, Disk usage) and input-output as a cost metrics (emit count, fail count, acknowledgement count) are some of the widely selected features on most state-of-the-art systems. A data-center system such as IBM Cloud Private [18], reports the performance of worker nodes to the master node in terms of CPUs, GPUs usage, and overall RAM utilization. Moreover, auto-scaling of running application totally depends on consumption of these contemporary components. Poggi et al. [19] also includes these system configuration metrics to report resource consumption based on the query to have a overall insight of cluster.

Data streaming performance prediction model

Regression algorithms are best candidate to perform prediction of any component in terms of latency. Since this problem is dealing with densely populated high-dimensional input data but only having continuous attributes, which makes it appropriate to apply parametric ridge regularization regression algorithms. The non-parametric regression algorithm such as support vector regression algorithms (ϵ-SVR, *nu*-SVR) also be the good candidate as it has less memory overhead in comparison with ridge regularization regression algorithm (label them as MSL in rest of the paper) but it

Table 1 Comparison of regression algorithm with respect to three evaluation metrics (with cross validation k = 10) performed on (i) Cluster-I, (ii) Cluster-II and (iii) Cluster-I and II

Regressions	Cluster	MSE	RMSE	MAE	R^2
Lasso regression	C-I	68.9297	8.0056	5.9667	0.9807
Lasso regression	C-II	58.4858	7.5675	5.3386	*0.963*
Lasso regression	*C-III*	*42.481*	*6.518*	*4.649*	0.99
Ridge regression	C-I	66.7715	8.1146	5.8574	0.9813
Ridge regression	C-II	55.2465	7.3488	5.2273	0.9663
Ridge regression	*C-III*	*40.433*	*6.359*	*4.546*	*0.991*
Elastic net regression	C-I	69.1028	8.0056	5.9757	0.9805
Elastic net regression	C-II	60.7171	7.6966	5.3623	0.9618
Elastic net regression	*C-III*	*42.578*	*6.525*	*4.693*	*0.99*
ϵ-SVR linear kernel	C-I	316.88	*8.1301*	13.7762	0.9113
ϵ-SVR linear kernel	C-II	298.6252	16.7423	13.3822	*0.813*
ϵ-SVR linear kernel	*C-III*	*126.669*	11.255	*9.044*	0.971
nu-SVR linear kernel	C-I	227.4365	17.0608	11.7427	0.9331
nu-SVR linear kernel	C-II	214.9223	14.2851	10.9677	*0.8661*
nu-SVR linear kernel	*C-III*	*132.333*	*11.502*	*9.289*	0.969

C-I, Cluster-I; C-II, Cluster-II; C-III, Cluster-I and Cluster-II

Italic values indicate the significance of various regression techniques (a minimum value of error within clusters)

outperforms the SVR with respect to all the three validation metrics as shown in Table 1 for each distinct types of cluster discussed in "Environmental setup" section. We compare the performance of these regression models using various measures such as mean squares of the deviations, square root of the arithmetic means of imperfection measure, mean absolute error and coefficient of determination as described in Table 1. We also used cross-validation (k = 10) for various selections of kernel and the result depicts that Ridge regularization regression algorithm outperforms the linear kernel, the polynomial kernel, the radial basis function kernel and the sigmoid kernel for each ϵ-SVR and *nu*-SVR.

To compare the efficiency of proposed model (label as *DKL* in rest of paper), we use well-studied technique for regression problem used in most performance tuning module of Distributed Database Management System. The dimensions of all the dynamic metrics are reduced using state-of-the-art dimensionality reduction technique called Factor Analysis. It transform the high dimensional stream processing systems dynamic metric data into lower dimensional data. Based on our experiments, we found that only the initial factors are most significant for our prediction framework due to existence of major influenced metrics distribution. To find out the highly influential metrics, we use k-means clustering algorithm to cluster this lower dimensional data using each row as its feature metrics and, keep a single metric from each cluster (one which were nearest to the centroid of a cluster). Finally, we use Gaussian processes regression to recommend performance of data streams with help of top k dynamic metrics of stream data processing system.

Metric pipeline

The function of metrics pipeline is to keep running as multiple threads throughout the live compute nodes in the cluster. When tuples arrives from one component of topology, the entire dynamic metrics are recorded with a encoded meta-attribute ID and timestamp. Then the tuple will be processed and stream the aggregate metrics to adjoint component, while entire dynamic metrics are again recorded by the metric pipeline along timestamp including tuple ID.

$$t_{thread} = \max_{1 \leq j \leq t} (n \times t_j) \tag{6}$$

Suppose there are total of t topologies and at any given time, maximum n nodes are consumed up by topology t_{t1}. Therefore, during a time frame t_{thread} are maximum threads running as shown in Eq. 6.

Data enrichment

The *Data Enrichment* converge all tuples record into new record having all dynamic metrics with help of unique tuple ID and timestamp. After concatenation, all the missing values are replaced with mean of entire column since all dynamic metrics record contains sparse data and distributed data storage termed as Data Grid.

Experimental evaluation

In this section, we provide the brief overview of benchmark suite used in the evaluation of proposed framework followed by a glimpse of experimental setups. The benchmark topologies form a data pipeline using open-source distributed pub-sub messaging system Apache Pulsar [11] to consume text streams generated by parallel synthetic data load generator. The input streams are the tuples which are generated using Alice's Adventures in Wonderland[2] text file and the spout consume the data streams later emits into topology through subscription to pulsar topic.

Environmental setup

We perform the evaluation of the proposed performance modeling framework over Apache Incubator Heron 0.17.1 release [4] on top of Centos Linux 7. Entire methodologies are evaluated based on their performance on two different clusters having heterogeneous architectures. The Cluster 1 consists of 19 Intel Xeon E5-2683 v4 nodes running at 2.10 GHz. Each compute cores has 128 GB RAM and 64 cores (2 sockets $\times$ 16 cores each with SMTP value of 2). The Cluster 2 consists of 12 many integrated core nodes named as Intel Knights Landing Xeon Phi 7250 running at 1.4 GHz where each KNL node composed of 64 GB and 16 GB MCDRAM having 72 cores each. Each computing node from the cluster is interconnected with NL-SAS Directed attached storage of 108 TB along with 100 Gbps Omni-Path fabric interconnect network for data and 1 Gbps Ethernet network for management which helps in maintaining overall cluster stability. Five different domain representative topologies were implemented and deployed. The

[2] http://www.gutenberg.org/files/11/11-pdf.pdf.

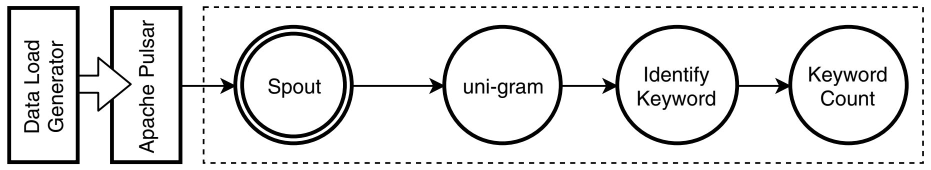

Fig. 3 Micro-Benchmark. A keyword search topology

latency including throughput is measured during changes in number of parallel task and system performance.

Benchmark suite

Grep count directed acyclic graph (GC-DAG)

The four stage topology widely known as a application of MapReduce. Their structure is alike to a chain of components comprised of three bolts and one spout as shown in Fig. 3 and it operates at the level of sentences with a maximum length of 119 characters. The spout is connected with the *uni-gram bolt* which convert texts into uni-gram tokens. These tokens are fed into *Identify Keyword* bolt which looks for keyword defined by a user. The *Keyword Count* bolt make a count for the presence of uni-gram in a tuple and store all results into a single file database[3] using field grouping. In the experiments, the processing logic components was set to have thirteen stream managers comprising of one spout executors, four uni-gram bolt executors, four Identify Keyword executors and four Keyword Count executors.

GEneral matrix to matrix multiplication directed acyclic graph (GEMM-DAG)

The three-stage topology structure is alike to a chain of components comprised of three parallel bolts and one spout as shown in Fig. 4. To evaluate our framework, micro-benchmark topology operates at the level of sentences with a maximum length of 20 words. The purpose of this topology is to have a performance profile of data processing platform during computation of CPU intensive operation on data stream along with deep learning model operations and matrix multiplication are the best candidate in such domain. Since performance of such operation varies due to dependency on size of matrices and kernel implementation along with the type of bound which can be computed, bandwidth and occupancy bound. Surprisingly, the way these models are utilized in practice are diverse as the optimization space for hardware and software targeting deep learning is large and underspecified [20–22]. The spout is connected with two parallel bolts named as *Matrix A* and *Matrix B* which generates the matrix based on tuples received from the spout. These bolts generates a sparse matrix of certain user defined size based on the tuples they received from bolts and the presence of common words among two tuples. These matrices are fed into the *GEMM* bolt which performs the multiplication operation after receiving matrices from both tuples. The results are stored

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

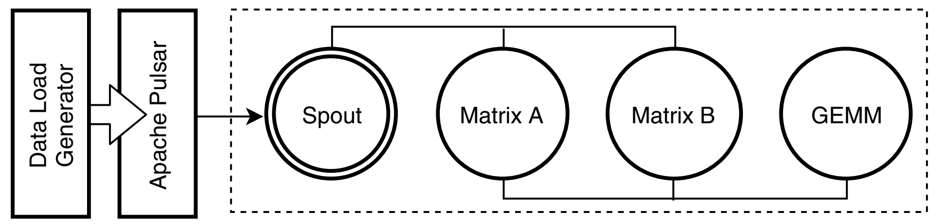

Fig. 4 Micro-Benchmark. A GEneral matrix to matrix multiplication topology

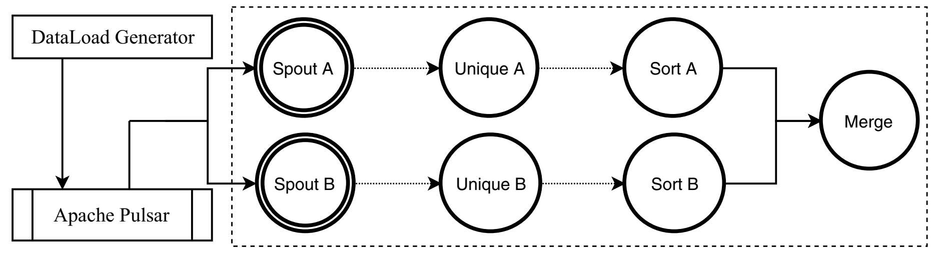

Fig. 5 Micro-Benchmark. A unique sort topology

into a single file database (see footnote 3) using shuffle grouping and tumbling window. In the experiments, the processing logic components configured to have twelve stream managers comprising of one spout executor, six matrix A executors, six matrix B executors and ten GEMM executors each perform operation on matrix of size 1048 with value of BLAS $\alpha \in 1$ and BLAS $\beta \in 0$.

Unique sort acyclic graph (US-DAG)

The four stage topology structure is alike to a chain of components having five bolts including two adjacent parallel bolts and two parallel spouts as shown in Fig. 5. To evaluate our framework, micro-benchmark topology operates at the level of sentences with a maximum length of 30 words. The purpose of this topology is to have a performance profile of data processing platform during computation of memory intensive operation on data streams and sorting algorithms are the best candidate for memory intensive computation. Since performance of such operation varies due to dependencies on size of inputs and kernel implementation along with the type of stableness it achieve. The spout is connected with bolt named as *Unique A* which transforms into tuples containing unique words based on tuples received from the spout. These transformed are fed into bolt named as *Sort A* which sort these tuples using standard merge sort algorithms which itself has $\mathcal{O}(n)$ space complexity. Lastly, sorted tuples are merged into a single stream and stored into a single file database (see footnote 3) using shuffle grouping. The experiments conducted with 16 stream managers comprising of one spout executor, twenty two unique A executors, twelve sort A executors and five merge bolt executors.

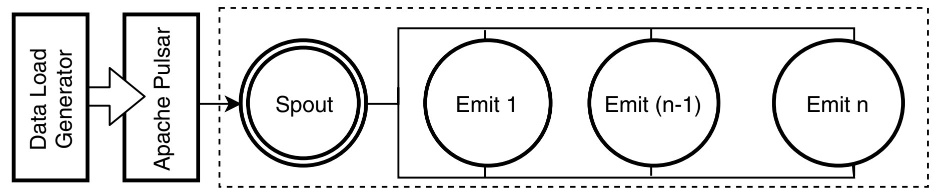

Fig. 6 Micro-Benchmark. A speed of light compute topology

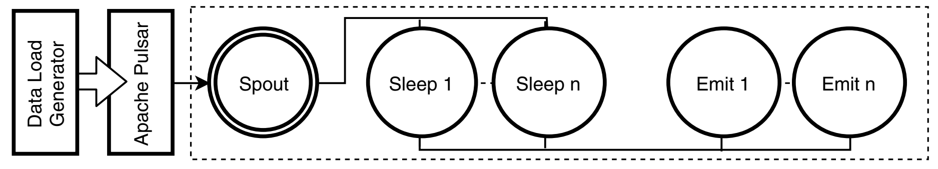

Fig. 7 Micro-Benchmark. A speed of light compute sleep topology

Speed of light compute directed acyclic graph (SOL-DAG)

The two stage topology has a chain-like structures with a group of bolts and a spout as shown in Fig. 6. To evaluate our framework, micro-benchmark topology operates at the level of sentences with a user-defined length of words. The purpose of this topology is to have a performance profile of data processing platform during computation of network intensive operations on data streams and fast tuples consumption algorithms are the best candidate for network intensive computation with varying message size. Since performance of such algorithms varies due to presence of number of bolts, message size and bandwidth used to interconnect these components running on distributed compute nodes. The spout is connected with the user-defined count of bolts termed as *Emit 1* and reach till *Emit* $(n - 1)$. The goal of this topology is to have a performance trace of network therefore we try to keep as minimum computation as can. Lastly, emitted counts are stored into a single file database (see footnote 3) using shuffle grouping. The experiments conducted with 19 stream managers comprising of one spout executor, sixty emit *n* executors.

Speed of light sleep directed acyclic graph (SOLS-DAG)

The three-stage topology structure is alike to a chain of components having two groups of bolts and spout as shown in Fig. 7. To evaluate our framework, micro-benchmark topology operates at the level of sentences with a fixed length of words. The purpose of this topology is to have a performance profile of data processing platform during computation of scheduler intensive operation on data streams and placing a bolt into an idle state for fixed quanta of time, which makes it a best candidate for scheduler intensive computations. Most state-of-the-art big data systems scheduler has the capability to integrate with existing version of heron and provide state-of-the-art computation solutions but how effectively it copes with the system calls at runtime totally depends on the scheduler being used. Although there is a assurance of minimum time duration but there is not strict assurance in executing the contemporary thread immediately. How these knobs are utilized in practice can be diverse since the optimization space is large and underspecified because these assurance is dependent on thread priorities and scheduler's decision. The spout is connected with the user-defined count of bolts termed as *Sleep 1* and reach till *Sleep (n − 1)*. Tuples from these bolts are fed into groups of bolts termed as *Emit 1* and reach till *Emit (n − 1)*. The goal of this topology is to have performance traces of scheduler which inspire us keep to keep minimum processing logic at the contemporary components. Lastly, emitted counts are stored into a single file database (see footnote 3) using shuffle grouping. In the experiments, heron cluster is configured with 16 stream managers comprised with one spout, ten sleep bolt executor with a parallelism of eight and ten emit bolt with a parallelism count one.

Results and inferences

In this section, we describe the brief experimental results of the five benchmark topologies in "Experimental evaluation" section. These are based on three commonly used metrics *Average Accuracy Rate* over a fixed quanta of time (here, 60 s).

$$Accuracy = 100 - absolute\left[\frac{Estimated - Actual}{Actual}\right] * 100 \tag{7}$$

Average error rate over a fixed quanta of time (here, 120 s) and evaluated using (100—accuracy) and *Overall Execution Latency* comprised of default and estimated execution latency of processing logic units over 20 min window frame. In the corresponding Figs. 8, 9, 10, 11 and 12 , "PL-SNumber", "SNumber–PNumber", "PNumber-Sq" and "Overall" resembles to the predicted accuracy, predicted error rate of tuple transmission latency among pulsar processing component and spout, spout and bolt, bolt and bolt, bolt and SQLite edging bolt after stabilization respectively. From these experimental norms, we can make the following observations.

GC-DAG topology

The experimental conclusion of the contemporary graph is delineated in Fig. 8. The average prediction accuracy rate of individual processing logic components are shown in Fig. 8a that varies from 99.91% (among the source component and uni-gram component) to 88.45% (at keyWord Count component) for MSL model. The performance assessment of contemporary models would be interesting in presence of unpredictable workload variation. To represent dynamic behavior we forcefully ingest skewness in the processing of user-defined components by restricting the parallelism count to be four. Due to dynamic variations in process unit metrics, the available metrics are far enough to cover all possible values; thus it reduces the predictive accuracy of topology to 90.90%, which is slightly higher than individual accuracy rates. Surprisingly, for model DKL prediction accuracy rate of individual processing logic components varies from 99.65% (among the source component and uni-gram component)

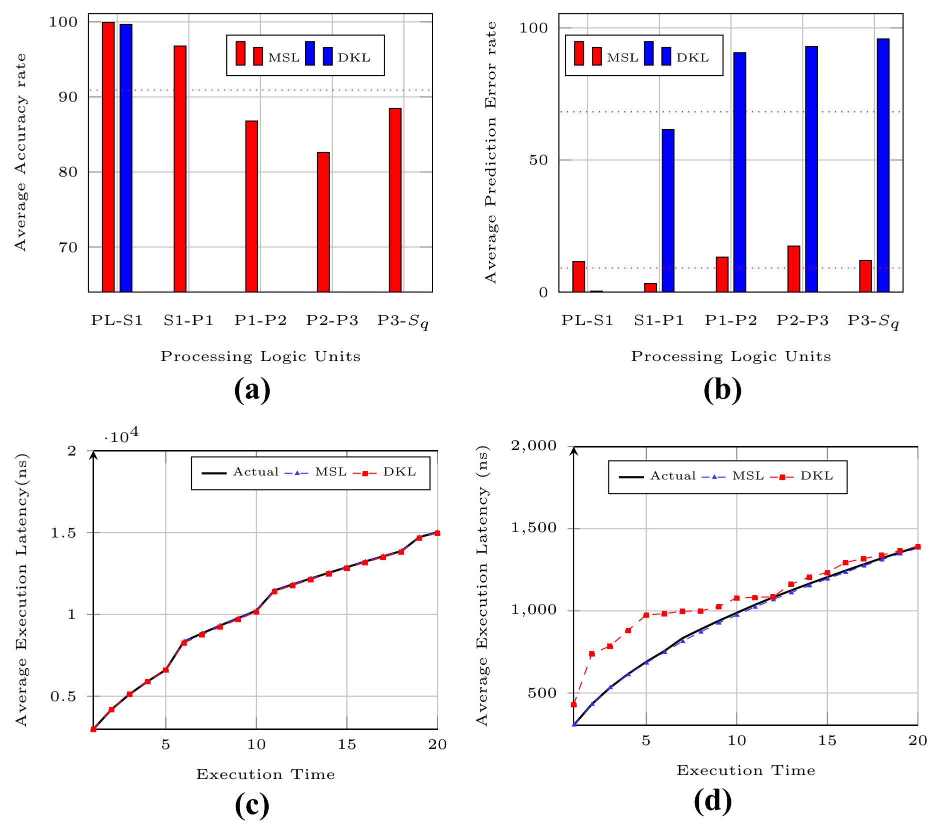

Fig. 8 Performance behavior of a GC-DAG topology over three metrics, **a** Represents the average accuracy rate corresponding to the groups of various processing logic units. **b** Represents the average prediction error rate corresponding to the groups of various processing logic units. **c** Represents the average execution latency of various spouts for a regular execution time frame of 20 minutes. **d** Represents the average execution latency of various bolts for a regular execution time frame of 20 minutes

to 4.585% (at keyWord count edging component). The presence of sparse attributed metrics leads to 29.89% reduction in prediction accuracy which is slightly lesser than prediction accuracy of individual processing component. Later, based on the experimental conclusion we found that discussed accuracy is much lower than overall accuracy achieved through MSL performance model. The average prediction error rates vary from 11.54% (among the source and uni-gram component) to 11.96% (at keyWord count component) and 0.348% (among the source and uni-gram component) to 95.84% (at keyWord count edging component) for MSL and DKL models respectively as shown in Fig. 8b. Even though DKL prediction model achieves an average accuracy of 29.89% but overall how it actually performs when its estimated latencies are compared with default dynamic latencies along the latencies estimated with MSL model are shown in Fig. 8c, d for a regular time frame of 20 min for spouts and bolts components respectively. Moreover, the default normalized dynamic execution latencies of bolts is much lower than the normalized estimated execution latencies of DKL prediction model as shown in Fig. 8d. However, as shown in Fig. 8c there is no significant

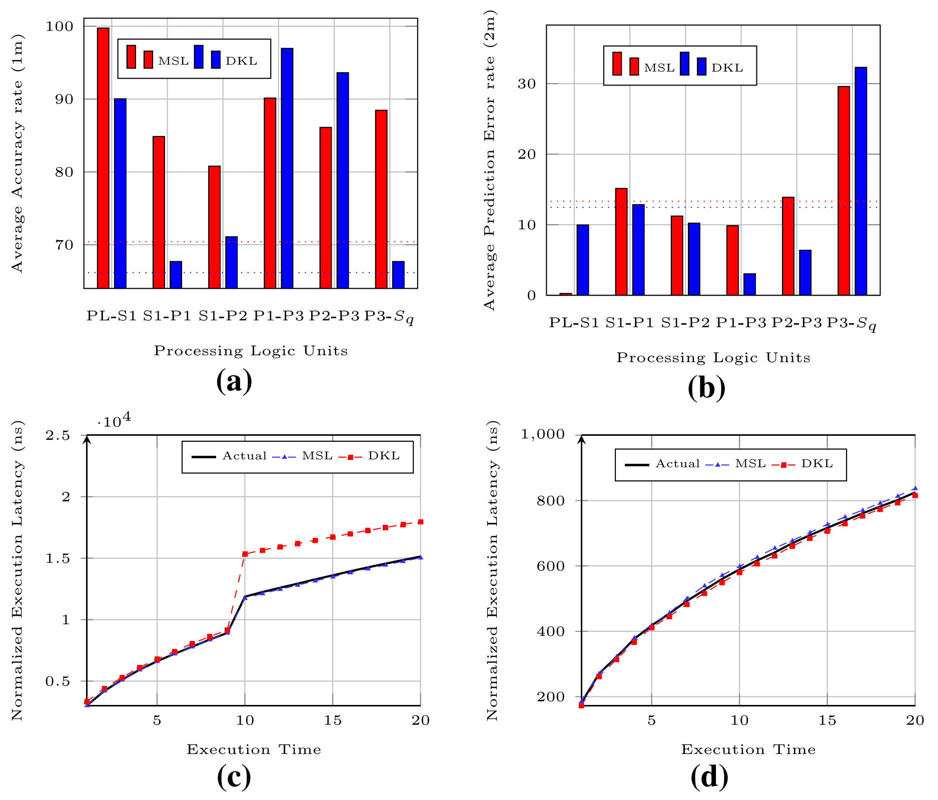

Fig. 9 Performance behavior of a GEMM-DAG topology over three metrics, **a** Represents the average accuracy rate corresponding to the groups of various processing logic units. **b** Represents the average prediction error rate corresponding to the groups of various processing logic units. **c** Represents the average execution latency of various spouts for a regular execution time frame of 20 minutes. **d** Represents the average execution latency of various bolts for a regular execution time frame of 20 minutes

difference among default and estimated normalized dynamic execution latencies of spouts.

GEMM-DAG topology

The experimental conclusion of the contemporary graph is delineated in Fig. 9. The average prediction accuracy rate of individual processing logic components are shown in Fig. 9a that varies from 99.74% (among the source component and matrix component) to 86.10% (at GEMM component) for MSL model. The performance assessment of contemporary models would be interesting in presence of unpredictable workload variation. To represent dynamic behavior we forcefully ingest skewness in the processing of user-defined components by restricting the parallelism count to be six. Due to dynamic variations in process unit metrics, the available metrics are far enough to cover all possible values; thus it reduces the predictive accuracy of topology to 70.41%, which is slightly higher than individual accuracy rates. Surprisingly, for model DKL prediction accuracy rate of individual processing logic components varies from 90.03% (among the source component and matrix component) to 67.69% (at GEMM edging component). The presence of sparse attributed metrics leads to

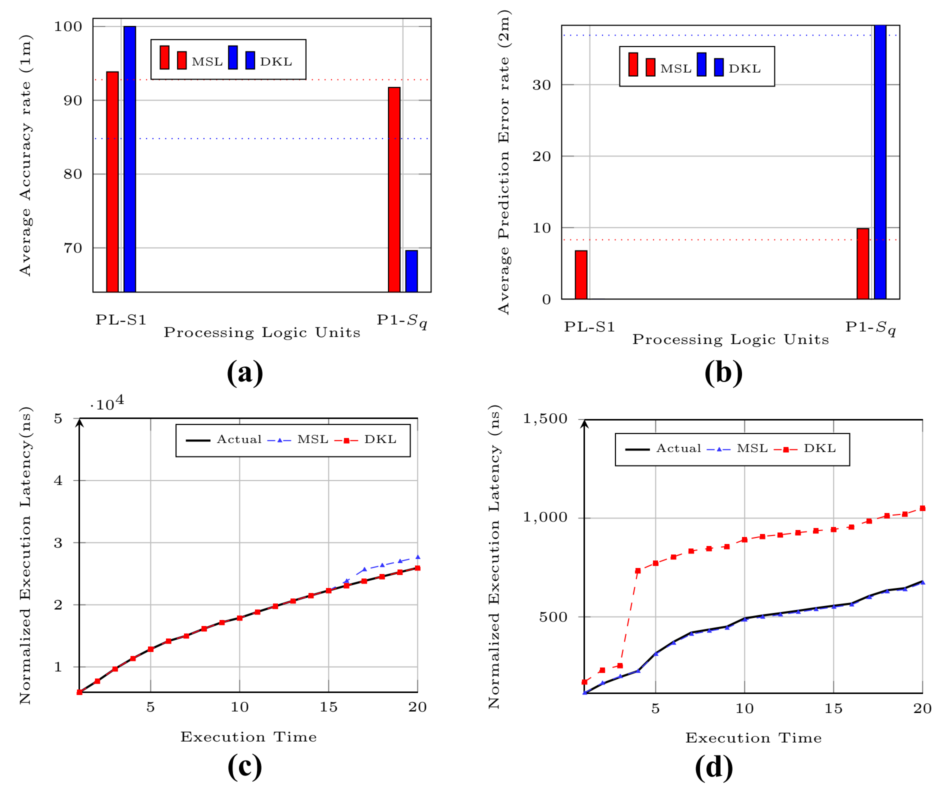

Fig. 10 Performance behavior of a SOL-DAG topology over three metrics, **a** Represents the average accuracy rate corresponding to the groups of various processing logic units. **b** Represents the average prediction error rate corresponding to the groups of various processing logic units. **c** Represents the average execution latency of various spouts for a regular execution time frame of 20 minutes. **d** Represents the average execution latency of various bolts for a regular execution time frame of 20 minutes

66.18% reduction in prediction accuracy which is slightly lesser than prediction accuracy of individual processing component. Later, based on the experimental conclusion we found that discussed accuracy is much lower than overall accuracy achieved through MSL performance model. The average prediction error rates vary from 0.25% (among the source and matrix component) to 29.58% (at GEMM component) and 9.96% (among the source and matrix component) to 32.30% (at GEMM edging component) for MSL and DKL models respectively as shown in Fig. 9b. Even though DKL prediction model achieves an average accuracy of 66.181 percent but overall how it actually performs when its estimated latencies are compared with default dynamic latencies along the latencies estimated with MSL model are shown in Fig. 9c, d for a regular time frame of 20 min for spouts and bolts respectively. Moreover, the default normalized dynamic execution latencies of spouts is much lower than the normalized estimated execution latencies of DKL prediction model as shown in Fig. 9c. However, as shown in Fig. 9d there is no significant difference among default and estimated normalized dynamic execution latencies of bolts.

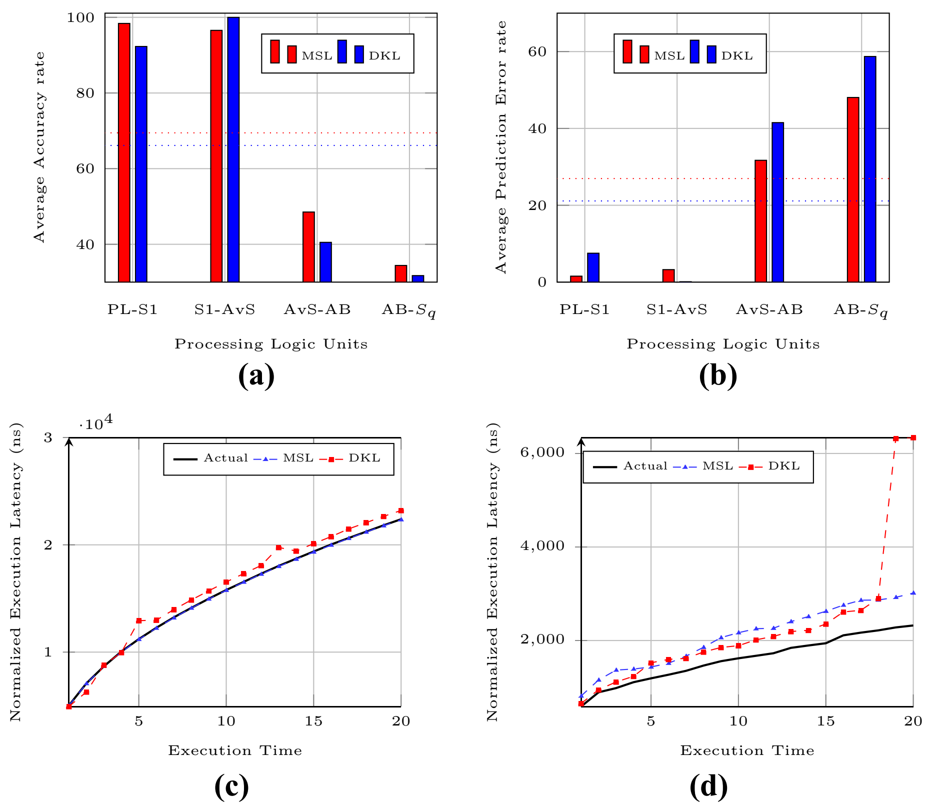

Fig. 11 Performance behavior of a SOLS-DAG topology over three metrics, **a** Represents the average accuracy rate corresponding to the groups of various processing logic units. **b** Represents the average prediction error rate corresponding to the groups of various processing logic units. **c** Represents the average execution latency of various spouts for a regular execution time frame of 20 minutes. **d** Represents the average execution latency of various bolts for a regular execution time frame of 20 minutes

SOL-DAG topology

The experimental conclusion of the contemporary graph is delineated in Fig. 10. The average prediction accuracy rate of individual processing logic components are shown in Fig. 10a that varies from 93.85% (among the source component and emit component) to 91.73% (at emit component) for MSL model. The performance assessment of contemporary models would be interesting in presence of unpredictable workload variation. To represent dynamic behavior we forcefully ingest skewness in the processing of user-defined components by restricting the parallelism count to be one. Due to dynamic variations in process unit metrics, the available metrics are far enough to cover all possible values; thus it reduces the predictive accuracy of topology to 92.79%, which is slightly lower than individual accuracy rates. Surprisingly, for model DKL prediction accuracy rate of individual processing logic components varies from 99.99% (among the source component and emit component) to 69.63% (at emit edging component). The presence of sparse attributed metrics leads to 84.81% reduction in prediction accuracy which is slightly lesser than prediction accuracy of individual processing component. The average prediction error rates vary from 6.76% (among the source and emit component) to 9.85% (at emit component) and 0.01% (among the source and emit component) to 73.81% (at

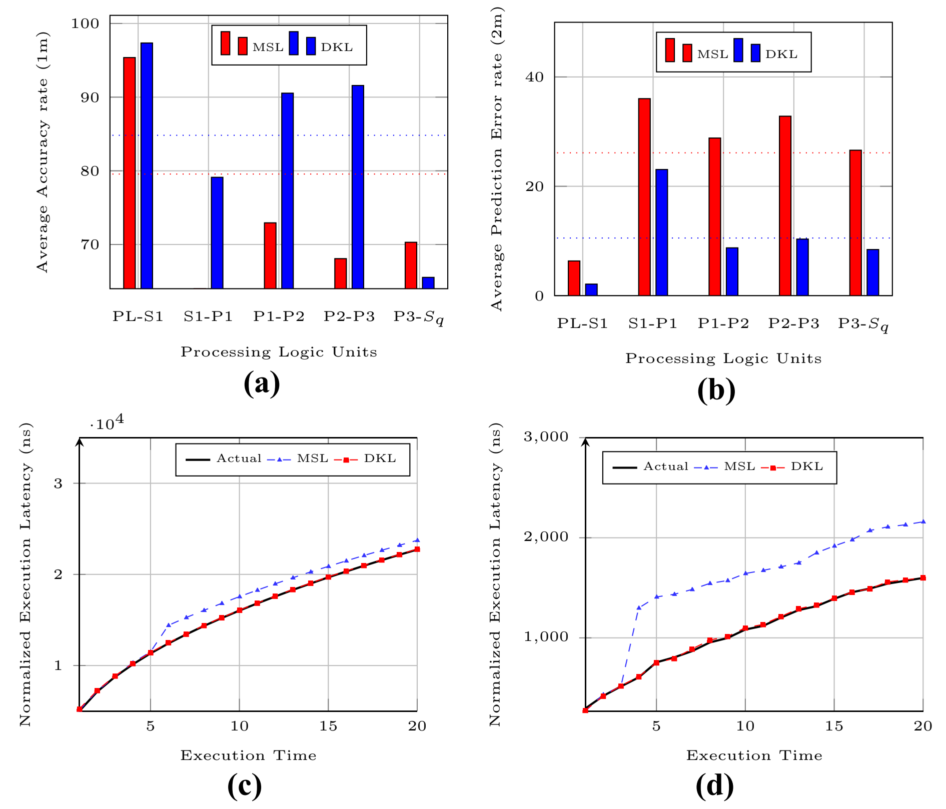

Fig. 12 Performance behavior of a US-DAG topology over three metrics, **a** Represents the average accuracy rate corresponding to the groups of various processing logic units. **b** Represents the average prediction error rate corresponding to the groups of various processing logic units. **c** Represents the average execution latency of various spouts for a regular execution time frame of 20 minutes. **d** Represents the average execution latency of various bolts for a regular execution time frame of 20 minutes

emit edging component) for MSL and DKL models respectively as shown in Fig. 9b. The MSL model on an average perfectly estimate the default dynamic latencies but scenario are very different for model DKL as shown in Fig. 10c, d. Moreover, the default normalized dynamic execution latencies of spout is almost similar with DKL and MSL prediction model observations but there exist significant deviation in 15–20 min range which can be negligible since difference is very minute as shown in Fig. 10c. However, there exists significant difference as shown in Fig. 10d among default and estimated normalized dynamic execution latencies of bolts with DKL prediction model over a duration of 20 min.

SOLS-DAG topology

The experimental conclusion of the contemporary graph is delineated in Fig. 11. The average prediction accuracy rate of individual processing logic components are shown in Fig. 11a that varies from 93.38% (among the source component and sleep component) to 34.40% (at emit component) for MSL model. The performance assessment of contemporary models would be interesting in presence of unpredictable workload variation. To represent dynamic behavior we forcefully ingest skewness in the processing of user-defined components by restricting the parallelism count to be eight and one respectively. Due to dynamic variations in process unit metrics, the available metrics are far enough to cover all possible values; thus it reduces the predictive accuracy of topology to 69.46%, which is slightly lower than individual accuracy rates. Surprisingly, for model DKL prediction accuracy rate of individual processing logic components varies from 92.29% (among the source component and sleep component) to 31.72% (at emit edging component). The presence of sparse attributed metrics leads to 66.12% reduction in prediction accuracy which is slightly lesser than prediction accuracy of individual processing component. The average prediction error rates vary from 1.55% (among the source and sleep component) to 48.05% (at emit component) and 7.52% (among the source and emit component) to 58.72% (at emit edging component) for MSL and DKL models respectively as shown in Fig. 11b. The MSL model on an average perfectly estimate the default dynamic latencies but scenario are very different for model DKL as shown in Fig. 11c, d. Moreover, the default normalized dynamic execution latencies of spout is almost similar with DKL and MSL prediction model observations but there exist significant deviation in 0–20 min range which can't be negligible as shown in Fig. 11c with observations of DKL model. However, similar observation exists with bolts but there exists a huge variations among DKL and MSL prediction model comparing with default normalized execution latencies as shown in Fig. 11d over a range from 0 to 20 min.

US-DAG topology

The experimental conclusion of the contemporary graph is delineated in Fig. 12. The average prediction accuracy rate of individual processing logic components are shown in Fig. 12a that varies from 95.37% (among the source component and unique component) to 70.30% (at merge component) for MSL model. The performance assessment of contemporary models would be interesting in presence of unpredictable workload variation. To represent dynamic behavior we forcefully ingest skewness in the processing of user-defined components by restricting the parallelism count to be 12, 22 and 5 respectively. Due to dynamic variations in process unit metrics, the available metrics are far enough to cover all possible values; thus it reduces the predictive accuracy of topology to 79.56%, which is slightly higher than individual accuracy rates. Surprisingly, for model DKL prediction accuracy rate of individual processing logic components varies from 97.35% (among the source component and unique component) to 65.53% (at emit merge component). The presence of sparse attributed metrics leads to 84.82% reduction in prediction accuracy which is slightly lesser than prediction accuracy of individual processing component. The average prediction error rates vary from 6.34% (among the source and unique component) to 26.60% (at merge component) and 2.11% (among the source and unique component) to 8.43% (at merge edging component) for MSL and DKL models respectively as shown in Fig. 12b. The DKL model on an average perfectly estimate the default dynamic latencies but scenarios are very different for model MSL as shown in Fig. 12c, d. Moreover, the default normalized dynamic execution latencies of spouts is almost similar with MSL and DKL prediction model observations but there is small deviation among latencies in MSL observations comparing with default observations as shown in Fig. 12c. Unfortunately, there exists a huge variations among default normalized dynamic latencies and normalized latencies of MSL prediction model as shown in Fig. 12 d over a range from 2.5 to 20 min.

Related work

We provide a comprehensive review of the organization and the community-contributed application workload driven benchmark in this section. It also includes the brief overview of crucial research efforts made on several state-of-the-art performance prediction models in the domain of specialized and generalized big data processing systems.

Big data system benchmarking: inferences and metrics

Karimov et al. [23] measure the performance of three state-of-the-art stream processing framework over various events viz. *joins*, *aggregations*, *queries over increased windows size* including presence of *skewness in data*, *fluctuating workloads* and *back-pressure* with processing-time and event-time latency as a evaluation metric. Based on the experimental analysis, author suggest favourable conditions for various framework. Quan et al. [24] based on the conclusion with three distinct representatives systems proposed that performance response on hardware fluctuates with the change in application workload. This dynamic variation not only depends on characteristics of workload but also on the amount of data underlying computing node processing. They also inferred that there is strong performance relationship among the type of workload running on which computing node. Han et al. [25] inferred that efficiently benchmarking the huge data processing system provides accurate measuring of contemporary systems and five-span of these systems are the most prerequisite important to sustain huge precisions. They also introduce us with four further classifications about workload input data generation: ready-made datasets, a synthetic distribution based data generators, real-world data based data generators and hybrid generators and also about two sub-branches of benchmarks labeled as the micro and end-to-end benchmarks. Similarly, Han et al. [26] authors categorized the whole system level evaluation metric into *user-perceivable metrics*(how frequently it can collect streams) and *architecture metrics* (how frequently it can respond to a streams). Similarly, Veiga et al. [27] using various batch and iterative workloads evaluates overall performance on the basis of cluster size, Block size, Data Size, Interconnect Network, Nodes Configuration and execution time on data center system. Finally, Jia et al. [28] suggests benchmarking with single application will not be enough to consider various domains of workload.

Big data system benchmarking: performance prediction model

Gupta et al. [29] proposed a theoretical performance prediction model for big data processing system based on the new active data, historical data. And, with the help of machine learning algorithms, it generates the metadata for new active data based and determines the performance level of systems and configure the system based on the prediction using metadata. The major drawback of such model is that they are based on a static sampling of correlated data. Baru et al. [14] highlight the importance of application-level-data benchmarks that are striving to cover all aspects of the application from ingestion to analysis. Nikravesh et al. [30] provides the autonomic performance indicator to support scaling in a cloud environment. Authors periodically sample the values from time-series streams to correlate various workload pattern and accuracy of regression algorithms viz. multi-layer perceptron. However, based on the experimental analysis the performance model for stream processing framework exploiting multi-layer perceptron neural network not able to work well. The relevant variant to our work is proposed by Li et al. [31]. Their proposed methodology entirely depends upon reinforcement learning. The complexity of their approach grows linearly with an increment in searchable (action) space which makes it unfit for actual use and further discussion on predictable performance are described here [32].

Discussions and conclusions

In our experiments, one of the most important contribution is the characterize transformation of entire dynamic metrics into five distinct categorization named as *memory metrics*, *n-Verticals*, *Communication metrics*, *Computation metrics* and *scheduler metrics*. These feature classification helps in precise behavior of entire model as shown in "Results and inferences" section. Moreover, it can be inferred from experiment described in "SOL-DAG topology" section that the correct combination of number of stream managers, parallelism count and number or executors including state-of-the-art resources available in computing node that helps in maintaining entire topology health while processing of large streaming data and ends with resource requirement problem for topology. In this work, we study the problem of predicting the performance of data streams in distributed stream processing environment. We proposed a design, methodology and evaluation of performance prediction framework aiming at efficient, resource adaptive and high performance distributed streaming platform. The framework comprising of six functional modules that includes *metrics pipeline*, *data enrichment*, *metrics classification*, *data grid*, *trigger* and *prediction model*. The metrics classification module categorize the dynamic topology metrics into seven predefined class for better performance behavior analysis and data enrichment provides a solution for missing values if present. The data stream performance prediction module comprise of two models: MSL and DKL. The self-driven *MSL* model tries fit classified dynamic metrics using ridge regularization regression algorithm and fully-automated *DKL* model inherited from the state-of-the-art workload management module from distributed database management system for distributed stream processing systems. We implemented its base on Apache Heron (version 0.17.1) and evaluate it with proposed Streaming Benchmark Suite comprising five domain specific micro-benchmarking topologies. To evaluate the proposed methodologies, We forcefully ingest tuple skewness among the benchmarking topologies in order to setup ground truth for predictions. From experiments, we found that accuracy of predicting performance of data streams increased upto 80.62% from 66.36% along with reduction of error from 37.14 to 16.06%. This shows that our MSL model outperform the state-of-the-art DSK model and can be used in workload optimization, scheduling and resource management problems in distributed stream processing systems.

Authors' contributions
BG conducted the experiments, analyzed the results and drafted the document. AB provided valuable suggestions on improving the standards of the manuscript. Both authors read and approved the final manuscript.

Acknowledgements
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
Not applicable.

Funding
Not applicable.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 13 June 2018 Accepted: 26 December 2018
Published online: 07 January 2019

References

1. Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J, Bhagat N, Mittal S, Ryaboy D (2014) Storm@twitter. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data, SIGMOD '14. pp 147–156
2. Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache flink™: stream and batch processing in a single engine. IEEE Data Eng Bull 38(4):28–38
3. Akidau T, Balikov A, Bekiroğlu K, Chernyak S, Haberman J, Lax R, McVeety S, Mills D, Nordstrom P, Whittle S (2013) Millwheel: fault-tolerant stream processing at internet scale. Proc VLDB Endow 6(11):1033–1044
4. Apache heron git repository. https://github.com/apache/incubator-heron. Accessed 11 Apr 2018
5. Chun B-G, Condie T, Chen Y, Cho B, Chung A, Curino C, Douglas C, Interlandi M, Jeon B, Jeong JS, Lee G, Lee Y, Majestro T, Malkhi D, Matusevych S, Myers B, Mykhailova M, Narayanamurthy S, Noor J, Ramakrishnan R, Rao S, Sears R, Sezgin B, Um T, Wang J, Weimer M, Yang Y (2017) Apache reef: retainable evaluator execution framework. ACM Trans Comput Syst. 35(2):5
6. Apache aurora git repository. https://github.com/apache/aurora. Accessed 12 Mar 2018
7. Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J (2016) Borg, omega, and kubernetes. Commun ACM 59(5):50–57
8. Van Aken D, Pavlo A, Gordon G J, Zhang B (2017) Automatic database management system tuning through large-scale machine learning. In: Proceedings of the 2017 ACM international conference on management of data, SIGMOD 17. pp 1009-1024
9. Aboulnaga A, Babu S (2013) Workload management for big data analytics. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data, SIGMOD '13. pp 929–932
10. Curino C, Difallah D E, Douglas C, Krishnan S, Ramakrishnan R, Rao S (2014) Reservation-based scheduling: If you're late don't blame us!. In: Proceedings of the ACM symposium on cloud computing, SOCC '14. pp 1–14
11. Apache pulsar git repository. https://github.com/apache/pulsar. Accessed 11 Apr 2018
12. Kulkarni S, Bhagat N, Fu M, Kedigehalli V, Kellogg C, Mittal S, Patel J M, Ramasamy K, Taneja S (2015) Twitter heron: stream processing at scale. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, SIGMOD '15. pp 239–250
13. Arasu A, Babcock B, Babu S, Cieslewicz J, Datar M, Ito K, Motwani R, Srivastava U, Widom J (2016) STREAM: the stanford data stream management system. Springer. pp 317–336. https://doi.org/10.1007/978-3-540-28608-0_16
14. Baru C, Rabl T (2016) Application-level benchmarking of big data systems. Springer, New Delhi. pp 189–199. https://doi.org/10.1007/978-81-322-3628-3_10
15. Sahin S, Cao W, Zhang Q, Liu L (2016) Jvm configuration management and its performance impact for big data applications. In: IEEE international congress on big data (BigData Congress) 2016. pp 410–417. https://doi.org/10.1109/BigDataCongress.2016.64
16. Java garbage collection, oracle. https://docs.oracle.com/cd/E17802_01/j2se/j2se/1.5.0/jcp/beta1/apidiffs/java/lang/management/GarbageCollectorMBean.html. Accessed 12 Mar 2018
17. Destounis A, Paschos G S, Koutsopoulos I (2016) Streaming big data meets backpressure in distributed network computation. In: IEEE INFOCOM 2016—The 35th annual IEEE international conference on computer communications. pp 1–9. https://doi.org/10.1109/INFOCOM.2016.7524388
18. Ibm cloud private. https://www.ibm.com/blogs/cloud-computing/2017/10/what-is-ibm-cloud-private. Accessed 12 Mar 2018
19. Poggi N, Montero A, Carrera D (2018) Characterizing bigbench queries, hive, and spark in multi-cloud environments. In: Nambiar R, Poess M (eds) Performance evaluation and benchmarking for the analytics era. Springer, Cham, pp 55–74
20. Jia Y (2014) Learning semantic image representations at a large scale, Ph.D. thesis, EECS Department, University of California, Berkeley (May)
21. Hadjis S, Abuzaid F, Zhang C, Ré C (2015) Caffe con troll: shallow ideas to speed up deep learning. In: Proceedings of the fourth workshop on data analytics in the cloud, DanaC'15. pp 1–4
22. Deepbench, baidu research. https://svail.github.io/DeepBench. Accessed 12 Mar 2018
23. Karimov J, Rabl T, Katsifodimos A, Samarev R, Heiskanen H, Markl V (2018) Benchmarking distributed stream processing engines. CoRR abs/1802.08496.
24. Quan J, Shi Y, Zhao M, Yang W (2013) The implications from benchmarking three big data systems. In: Proceedings—2013 IEEE international conference on big data, big data , 2013. pp 31–38. https://doi.org/10.1109/BigData.2013.6691706
25. Han R, John LK, Zhan J (2018) Benchmarking big data systems: a review. IEEE Trans Serv Comp 11(3):580–597. https://doi.org/10.1109/TSC.2017.2730882

26. Han R, Jia Z, Gao W, Tian X, Wang L (2015) Benchmarking big data systems: state-of-the-art and future directions, CoRR abs/1506.01494. arXiv:1506.01494
27. Veiga J, Expósito RR, Pardo XC, Taboada GL, Tourifio J (2016) Performance evaluation of big data frameworks for large-scale data analytics. In: IEEE international conference on big data (Big Data) 2016. pp 424–431. https://doi.org/10.1109/BigData.2016.7840633
28. Jia Z, Wang L, Zhan J, Zhang L, Luo C (2013) Characterizing data analysis workloads in data centers. In: IEEE international symposium on workload characterization (IISWC) 2013. pp 66–76. https://doi.org/10.1109/IISWC.2013.6704671
29. Gupta S, Dominiak J, Marimadaiah S (2017) Using machine learning to predict big data environment performance, U.S Patent 2017-0140278 A1, 18 May
30. Nikravesh AY, Ajila SA, Lung C-H (2017) An autonomic prediction suite for cloud resource provisioning. J Cloud Comput 6(1):3. https://doi.org/10.1186/s13677-017-0073-4
31. Li T, Xu Z, Tang J, Wang Y (2018) Model-free control for distributed stream data processing using deep reinforcement learning. Proc VLDB Endow. 11(6):705–718
32. de Assuncao MD, da Silva Veith A, Buyya R (2018) Distributed data stream processing and edge computing: a survey on resource elasticity and futuredirections. J Netw Comput Appl 103:1–17. https://doi.org/10.1016/j.jnca.2017.12.001