TOWARD DESIGNING A DYNAMIC CPU CAP MANAGER FOR TIMELY DATA-FLOW (TDF) PLATFORM

M. Reza HoseinyFarahabady
The University of Sydney
Center for Distributed
& High Performance Computing,
School of IT, Australia
reza.hoseiny@sydney.edu.au

Saeed Bastani
Karlstad University,
Department of Computer Science,
Sweden
saeed.bastani@kau.se

Javid Taheri
Karlstad University,
Department of Computer Science,
Sweden
javid.taheri@kau.se

Albert Y. Zomaya
The University of Sydney,
Center for Distributed
& High Performance Computing,
School of IT, Australia
albert.zomaya@sydney.edu.au

Zahir Tari
RMIT University,
School of Science,
Australia,
zahir.tari@rmit.edu.au

Samee U. Khan
North Dakota State Uni.,
Dept. of Electrical
& Computer Engineering,
Fargo, ND, USA
samee.khan@ndsu.edu

SpringSim-HPC, 2018 April 23-26, Baltimore, MD, USA;

ABSTRACT

This work is one of the first attempts to address the problem of CPU resource allocation of a set of parallel *data-flow* applications that might have different quality of service (QoS) requirements. We observed that a static resource allocation schema might severely degrade the performance of some (if not all) data-flow applications as a result of fierce competition for obtaining CPU as the main shared resource in a cluster of server nodes. To allocate the best possible CPU cap, we propose a control-based solution that considers the degradation of performance caused by running concurrent data-flow processes. It uses the famous Kleinrock Formula for G/G/1/K queuing systems. The aim of the proposed dynamic solution is reduce the QoS violation incidents for all applications. It makes appropriate decisions by predicting the future entrance rate of data elements into each application. The performance of the proposed solution is bench-marked with the famous round robin algorithm, which is a classic but simple algorithm widely used as the resource allocation scheme in multiple commercial packages. The experimental results confirms that the proposed algorithm can reduce the latency of processing data records for applications by 48% compared to the round robin policy.

Keywords: Distributed data-flow processing engines, Scheduling and resource allocation algorithms, Quality of service

1 INTRODUCTION

Timely data-flow is the most recent programming model that supports application developers with a rich set of primitives to develop distributed and parallel applications that consist of iterative modules over streaming data sets (Murray et al. 2013). The demand for fast analysis over a large volume of such data led to the development of several batch/streaming data engines such as MapReduce (Dean and Ghemawat 2008), Apache Spark (Zaharia et al. 2010), CStream (Şahin 2015), Apache Storm (Apache Software Foundation 2016), IBM System S (Zou et al. 2010), Yahoo S4 (Neumeyer et al. 2010), SCOPE (Chaiken et al. 2008), and Microsoft Sonora (Yang et al. 2012). However, an important missing point of almost all above-mentioned systems is their intrinsic deficiency to brace *iterative* modules over incoming data (Wang 2013), (Clauss and Gustedt 2010), (Dudoladov et al. 2015). Supporting such a feature is almost inevitable for developing most of emerging applications in different domains such as learning algorithms and large scale graph-based data engines.

The main advantage of timely data-flow (TDF) paradigm is that it offers all three main benefits of batch and streaming computational models within the same system. Achieving such an ambitious goal is realized by supporting a mechanism that provides application developers for designing independent yet stateful iterative and incremental computational modules. In TDF paradigm, an application can be expressed as a collection of several directed (possibly *cyclic*) graphs (DCG) in which the set of graph vertex actually represents the parallel tasks of the application. Such tasks can exchange data elements along the directed edges of the DCG graph. Furthermore, each application's message is labelled with a non-conventional logical time-stamp (see (Murray et al. 2013)) for more details) to keep the state of computations.

On the other side, one of the most critical requirements for a data-flow processing engine that runs over a server farm of tens or even hundreds of server nodes is finding an effective way to fulfil the quality of service (QoS) requirements enforced by the application owners. In almost all existing practical solutions, a conventional resource allocation algorithm often picks a new server node only if the overall computing capacity of the current resources cannot cope with unexpected spikes in the incoming workload. Nevertheless, in practical scenarios in which the amount of computing capacity is limited, it is almost impossible to avoid violating of QoS requirements for all applications over the course of their execution. A conventional resource allocation strategy, such as round robin or best effort, which is not aware of QoS enforcement level might severely damage the performance level of those applications that belong to the highest QoS level when the

capacity of the available resources cannot cope with the incoming demand from all applications. In such cases, a *QoS-aware* resource allocation algorithm is needed to truly comply with the service-level objectives declared in SLA (service level agreement) while maximizing the overall performance of the system.

In this paper, we present an approach for the design as well as a prototype development of a close-loop controller as an elastic solution for allocating CPU shares in a TDF platform. The main key features of the proposed solution is that it considers QoS enforcement level when making resource allocation decisions. We use an ARIMA based prediction model to estimates the future rates for incoming data records per application. Then we apply a simple queue based system model proposed by Kleinrock in (Kleinrock 1976) to estimate the latency of each computational module in the next controlling interval using a history of past observations. In addition, the loop counter of each iteration module is estimated by employing a Monte Carlo sampling method, namely $\mathscr{AA}$ Algorithm, that uses the minimum possible number of measurements to estimate the unknown value of each loop counter. The experimental evaluation section presents the result of a series of tests that we have conducted as our evaluation study to test the performance of the proposed solution against the round robin policy with respect to both the average response time and the QoS violation rates. Particularly, the proposed solution outperforms the round-robin algorithm by an average improvement of 48% in the overall response time of processing data records, while it reduces the QoS violation incidents by a factor of 3.6 on scenarios that the available computing capacity of server nodes in the local cluster cannot cope with the total demand from all applications.

The rest of the article is organized as follows. In Section 2, we give the background knowledge and related work associated with the timely data-flow platform. Section 3 formally presents the design principle of the proposed resource allocation algorithm. In Section 4, we evaluate the performance of our solution through experiments on real systems. Section 5 briefly discusses the existing work. We then draw our conclusion in Section 6.

2 BACKGROUND

Timely data-flow can be considered as rather new yet evolutionary programming model for designing scalable and fault-tolerant distributed systems first introduced by Microsoft researches (Murray et al. 2013). The programming model can be employed by application developers for deploying parallel programs that runs continuously over a set of streaming data for further processing. Such a streaming processing has been recently receiving a lot of attention, particularly for dealing with the upcoming issues in processing in near real-time fashion.

The new cyclic data-flow model offers all three advantages of prevalent giant models for big-data processing in one system. Particularly, it offers (1) *high throughput* (for batch processing systems), (2) *low latency* (for stream processing engines), and (3) the ability to perform *iterative*, *stateful*, and *incremental* computations over incoming streaming data records. In fact, the complexities of combining such features (like nesting loops inside streaming contexts or keeping the state of the computations) in one system can be easily resolved by using timely data-flow model (Murray et al. 2013).

In this model, each message of any computational module is being attached with a logical time-stamp as a lightweight mechanism for the purpose of coordination among iterative and incremental processing modules. For the purpose of tracking the progress of the computations, the underlying platform keeps the set of point-stamps of all messages which are still in their execution progress. This helps each parallel worker to realize the set of pending messages that still needs to be delivered to each vertex node at any given time. It also can be used by every working threads to find out if some additional data records will be arrived in the future epochs (Chandu Thekkath 2017).

The directed graph of might consist of a a data-flow computation consists of a set of input, output, ingress, egress, feedback, and normal computational vertices. The input (output) vertex is declared to indicate a stream of data coming (emitting) from the an external producer (to an external consumer). Ingress, egress and feedback vertices are declared to identify a loop context (can be nested within other loop contexts). Such vertices are used to deliver the message in a correct order.

Controlling of the parallelism degree per each computational module can be determined by the application developer, while the underlying framework is responsible to deliver the set of right ordered messages to corresponding computational threads (Abadi et al. 2013). So, the resource allocation of a timely data-flow system can take advantage of such a fact and concurrently execute the same computational worker thread over different data streams. To reach the best resource performance, the resource manager has to find out the amount of CPU time assigned to each worker thread in a way to (1) balance the load among them and (2) to avoid any thread becoming a bottleneck. A fair round-robin policy that simply distribute the CPU capacity equally among worker threads cannot effectively cope with the temporal abnormal rise in the incoming requests.

3 THE PROPOSED RESOURCE ALLOCATION SCHEMA

We consider end-to-end response time as the main performance concern of the end-users. We use a model based on queuing theory to allocate/dismiss computing resources to each sub-component of the a timely data-flow application. We also use "control theory" principles to design a robust feedback controller for allocating CPU resources in a dynamic fashion. Our target platform is a cluster that hosts several data-flow applications belong to users with different QoS enforcements.

The key idea of the proposed solution is to leave the decision of allocating CPU resources to the execution time, when it can effectually prevent the issue of capacity bottleneck. The controller measures the following metrics to determine the online state of the system. (1) the incoming traffic rate of each computational module from external source, (2) the available CPU capacity of each host, and (3) the amount of QoS violation incidents occurred so far per each user. Whereas finding an exact relationship among different parts of the system under study might result in an accurate solution, employing a simpler linear difference equation that approximates such a relationship, like the one we use in this paper, is more common in practice. Employing a similar predictive mechanism to control the computing resources in a variety of evnet- and stream-based data processing platforms is not new and has been effectively used by several researchers in the past, such as (Casalicchio et al. 2013), (De Matteis and Mencagli 2016), (DeMatteis and Mencagli 2017), (HoseinyF. et al. 2017a).

We use one buffer per stateful computational module (C_i) to keep track of outstanding messages. By adjusting the percentage of the amount of CPU core cap to be assigned to the corresponding working thread of every computational module, we can respond to the temporal changes in their arrival rates while taking QoS enforcement level into account. Particularly, we approximate the end-to-end delays of each data-flow message by measuring the number of outstanding messages in each buffer at any given time. To this end, the controller uses a prediction tool to estimate the future rate of incoming data elements to each application over a finite-time horizon. Then, based on the current measurements and the predicated future states, the proposed controller takes the near-optimal CPU cap collocation decisions. We will show (in Section 4) that the given solution is robust despite the modelling errors.

We use auto regressive integrated moving average equation (ARIMA) model to estimate the input traffic *rate* of data records coming to each data-flow application within the next controlling interval. According to the ARIMA formula, the future values of a target random variable, λ_k, where k denote the interval index, is estimated based on previous observations of such a random variable as follows.

$$\hat{\lambda}_k = \varepsilon_k + \sum_{\ell=1\cdots h} \beta_\ell \lambda_{k-\ell} + \theta_\ell \varepsilon_{k-\ell} \tag{1}$$

, where ε's are i.i.d error values taken from a normal distribution with mean zero and a finite variance (*e.g.*, a white noise). The coefficients of β and θ are updated using the least-squares regression method right after realizing a new observation.

QoS Guarantee. For the sake of designing a QoS-aware resource allocation, we use the following equations to identify by how extent a resource allocation policy can satisfy the desirable performance metric from the user's perspective Let us assume that there are exactly Q different level of QoS classes from which an end-user can choose its desirable performance level. Each class $1 \leq q \leq Q$ is associated with two values of ω_q^* and $\mathscr{V}_q$ as the maximum acceptable average processing delay and an acceptable upper bound for the percentage of QoS violation incidents can occur for that class, respectively.

For example, let us assume that there are 3 different QoS classes, denoted by $q = 1..3$. Let us also assume that the associated upper bounds of each class is taken from $\mathscr{V}_{q=1..3} \in \{0.50, 0.80, 0.99\}$, where the third class (*i.e.*, $q = 3$) has the highest priority. So, the end-to-end response time of processing each data records belong to application from the third class can be just higher than ω_3^* only for 1% of the whole set of data records during any given period.

Controlling Algorithm. There are at least two key concepts in designing a feedback control for a computing system: (1) the measured output vector, as the system characteristic to be controlled as close as possible as to a desired value (it typically depends on the nature of workload being served); and (2) the control input vector, as the system variables that influence the system output. The input vector is manipulated either by the system administrator or an automatic controlling mechanism to modify the system output. The first step to designing a good feedback control is understanding how the input vector modifies the the measured system output. Further, it is important to realize that the system output value also depends on the system workload, that are normally unknown *a priori* or can change over the execution time (Hellerstein et al. 2004).

Besides to the control input and output vectors, there are other essential elements of a feedback control system as (i) the set-point trajectory, as the desired value of the control output vector, and (ii) the error vector, that is the difference between the set-point trajectory and the measured system output. The main duty of the automatic controller is to find a setting for the control inputs such that the system output follows the set-point trajectory as the reference point. The main advantage of using such a feedback control is that the system administrator just needs to specify the desired output value as the right set-point trajectory instead of direct steering of the input variables which requires substantial expertise set.

The aforementioned queuing model can be successfully applied to handle the issue of service-level guarantees. We regulate the end-to-end response time of each data-flow application by introducing a feedback controller for manipulating the service rate (*i.e.*, the CPU cap) of each application. The implicit objective of such a system is to maximize the number of processed data-flow requests subject to constraints on QoS enforcements (*i.e.*, response times). In our target system, the reference *set-point trajectory* is the desired response time according to QoS enforcements, and the control *input vector* is the CPU cap of all data-flow applications at each controlling interval. The *control error* needs to be calculated by subtracting the average response times from the reference trajectory.

At the beginning of each interval $k = 1, 2, \cdots$, the controlling algorithm accomplishes the following actions.

- It collects the number of pending messages in the buffer of each input module as well as the number of data records that is processed by each computational module during the previous interval.

- The controller predicts the arrival rate of messages per input module for the future step. Using such information, the controller can find out the response time of each application for the next step.
- The controller solve an optimization problem (described later) to find out the best possible CPU cap configuration per application, and then apply such an supposedly optimal solution.
- At time $k+1$, the controller observes the performance of the underlying system as the feedback loop and the above cycle is repeated.

We compute the CPU cap of each application based on the following model (which is obtained from queuing analysis) for the dynamic of a TDF platform. For the data records buffered in each computational unit, we assume a first-come, first-served scheduling policy. So, the data elements depart in the same order in which they arrive to the buffer of each module. We do not assume any further restriction on the inter-arrival time distributions. Let r_k denote the response time (experienced delay) of the k^{-th} data element, τ_k denote the time between the arrival of two consequent data elements at interval k, and s_k denote the service time needed to process such data element. Such variables are related based on the following equation as stated in (Kleinrock 1976).

$$r_{k+1} = (r_k - \tau_{k+1})^+ + s_{k+1} \tag{2}$$

, where $(x)^+$ denote the $\max\{0,x\}$.

If a module C_i resides within a loop context, then we should multiply the average delay obtained by Equation (2) with the mean value of the times that the loop context is executed, shown by $\bar{\eta}_{C_i}$. We use an estimation method based on Monte Carlo sampling algorithm to estimate the mean value of the loop variable for every computational modules. This algorithm, knowns as $\mathscr{A}\mathscr{A}$ Algorithm, is a fully polynomial randomized approximation scheme (FPRAS) that estimates the value of $\bar{\eta}$ by using the *minimum* possible number of measurements (Dagum et al. 2000).

After determining the desirable service time of each application for the CPU credit in the next controlling interval (*i.e.*, s_{k+1}), a central optimization module computes the CPU cap that must be allocated to each data-flow application . The optimization module formulates it as a *capital budgeting problem*, where the *finite budget* is the total amount of CPU share that is available on the server farm, and the *reward* function associated with each data-flow application is the multiplication of the QoS class that the application belongs to and its requested CPU demand. The solution can be found using a dynamic programming approach (for more details see (Powell 2007)).

4 EXPERIMENTAL EVALUATION

The effectiveness of the proposed approach is carried out with respect to QoS violation rate of data-flow applications. We assessed the proposed approach against round robin greedy algorithm that tries to assign CPU share of all accessible physical machines uniformly among data-flow applications.

We use a local cluster consisting of three machines with a total number of 12 logical cores. Each machine is installed with 8 GB of main memory and equipped with a 3.40 GHz Intel i3 CPU. The proposed controller is developed in C++ and Python 2.7 on the top of a modular open-source implementation of timely data-flow in Rust. The controller runs over a dedicated machine equipped with an Intel i7 2.3 GHz CPU with 16GB of RAM and a SSD drive. It can be equally applied to other implementation of timely data-flow, such as Microsoft .NET Naiad system (Microsoft Naiad 2017).

We created $n = \{50, 100, 150\}$ data-flow applications. Each application has four computational modules as a simple loop. Each module runs a dummy script that its running time varies from 0.1 second to 5.0

second with an average of 2.2 second. We also fixed the number of QoS classes to three ($|Q| = 3$) and randomly assign each application to one of the QoS classes. We bind the first computational module of each application to an external emitter which its generation rate is taken from a Poisson process with $\lambda \in [0.5, 2]$, where λ represents the average number of data records generated per 0.1 seconds. The controlling interval length is chosen to be 1 second.

Figure 1 depicts the average response time of the highest QoS class achieved by the proposed controller compared to the one from RR policy when the number of applications increases from 50 to 150 (x axis). The proposed algorithm can reduce the QoS violation incidents (Figure 2) by a factor of 3.6 compared to the RR heuristic on average, too. The experiment also confirms the need for a controlling mechanism in a environment where applications with different QoS levels share the CPU capacity. The main reason is that the static schema equally distributes the CPU shares among applications. Such a fair decision causes a severe QoS violations for applications of higher QoS classes particularity if a burst period exists in which some applications from lowest QoS class consume as much CPU capacity as they need without any restrictions.

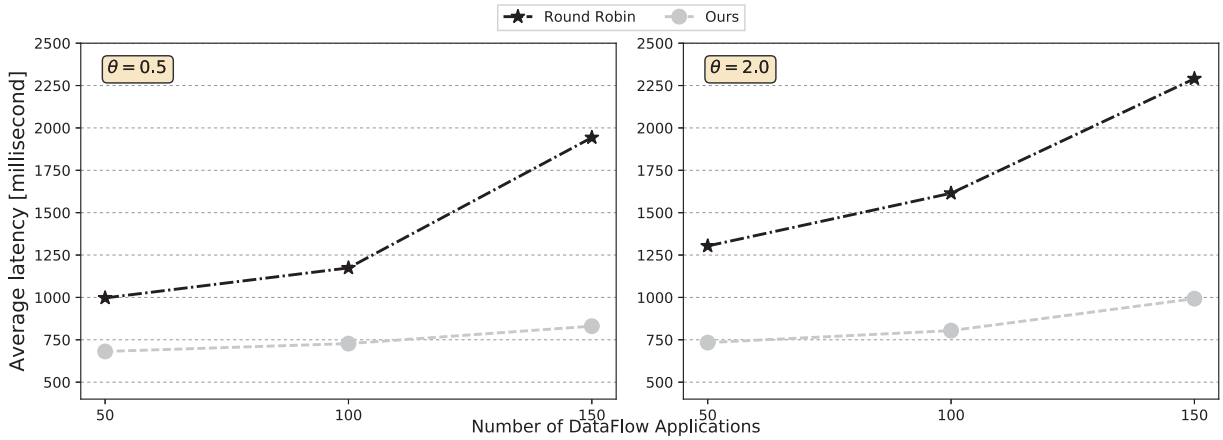

Figure 1: Improvement in average response time (latency) achieved by the proposed controller compared to the static round robin allocation schema as seen by applications belong to the highest priority QoS class.

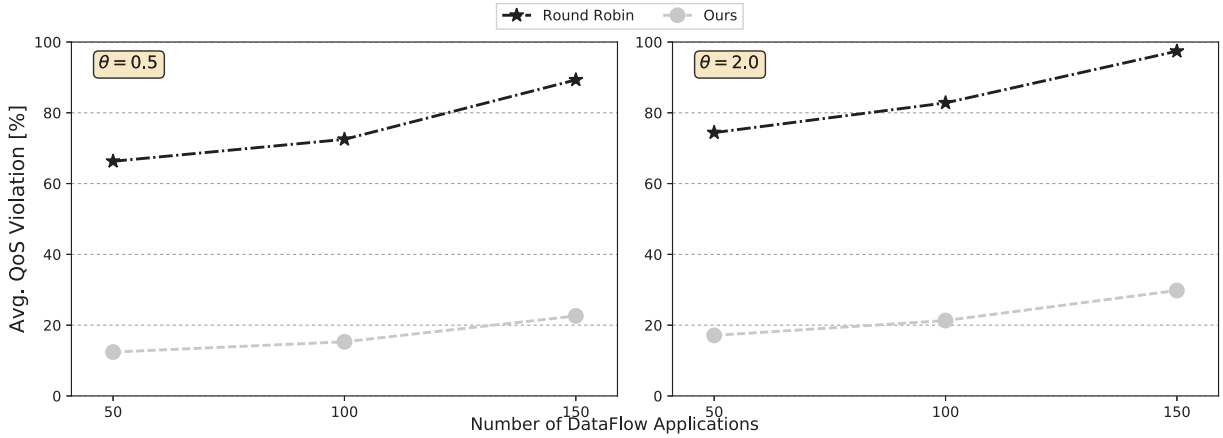

Figure 2: The percentage of the QoS violation incidents achieved by the proposed controller compared to the static round robin allocation algorithm as seen by applications belong to the highest priority QoS class as the number of TDF applications varies.

The running time of the proposed algorithm to find an optimal solution is negligible (less than 0.01 second) for a scenario with 150 data-flow applications and 3 machines. The reason is that we exploit a dynamic programming schema which can quickly find the optimal solution for the capital budget problem.

We also performed sensitivity analysis of the proposed controller against occurrence of errors in parameter prediction phase. We deliberately injected a prediction error to the generation rate parameter ranging from 10% to 90%, and then collected the relative influence of such an error on the output of the system. To this end, we use the sensitivity coefficient parameter (ψ) as introduced in (HoseinyF. et al. 2017b) to find out how much an output vector varies if there is an estimation error of $\varepsilon_{\mathbf{x}}$ in the input vector $\mathbf{x}$. More precisely,

$$\psi_{\varepsilon,\mathbf{z}} = \frac{|\mathbf{z}(\mathbf{x}) - \mathbf{z}(\mathbf{x} \pm \varepsilon)|}{|\mathbf{z}(\mathbf{x})|}$$

The experimental results confirmed that even an error of 90% in the prediction model has a negligible influence in the solution quality (up to 30% for delay and 13% for the QoS violation avoidance rate). The result showed that the proposed controller is robust enough against the accuracy of the prediction model to be used in real circumstances.

5 RELATED WORK

Several data-flow platforms have been used in different big data applications where their algorithms show iterative nature. Naiad (Murray et al. 2013) has been introduced as one of the earliest attempt to design a distributed system for running parallel and iterative operations over either batch or streaming data records. C-Stream (Şahin 2015) is another elastic streaming processing engine implemented in C++11. Unlike Naiad, operator in C-Stream needs to explicitly request incoming data-tuples from the corresponding ports.

Apache Spark (Zaharia et al. 2010) is a fast, in-memory data processing engine to execute iterative algorithms over the streaming data-sets. It probably has the most similarity with timely data-flow paradigm when compared to other existing frameworks. On the other hand, Cilk (Blumofe et al. 1996) is the most famous work-stealing schedulers for multi-threaded parallelism, where each task produces additional workers by spawning more tasks. In the context of timely data-flow, such a work stealing from other threads is not a straightforward concept to be implemented and may increase interference among threads.

Many existing resource allocation strategies, such as (Li et al. 2014), (Huang et al. 2016), (Padala et al. 2009), (Xiaoyong et al. 2011), (Ballani et al. 2011), (Shen et al. 2011), (Li et al. 2014), (Huang et al. 2016), manage resources based on OS level metrics, such as per core utilization, I/O capacities, and energy usage of resources while ignore the negative performance caused by interference at the shared resources, like LLC or memory bandwidth.

Using a control mechanism is not new in computing systems, such as the work in (Mencagli et al. 2014), (Mencagli 2016), (Abdelwahed et al. 2009), (Mencagli et al. 2014), (Mencagli 2016), and (Padala et al. 2009). Most of these works proposed a multi-input, multi-output (MIMO) resource controller that automatically adapts to dynamic changes in a shared infrastructure. Such models try to estimate the complex relationship between the application performance and the resource allocation, and adjusts the embedded model by measuring the clients' response time. While there are similarities between the proposed solution with previous controllers, our solution responds to the degraded performance level by measuring the number of waiting messages and then applying a more accurate queuing based formula to estimate the response time of each application.

6 CONCLUSIONS

We presented a resource allocation strategy for timely data-flow in a shared platform. Timely data-flow is a powerful and general-purpose programming abstraction for creating iterative and streaming computational components. Our aim is to design an elastic resource allocation strategy that continually monitors the important performance metrics of the underlying system and assigns the amount of CPU to DFT applications that belong to different QoS level. The effectiveness of the proposed controller has demonstrated an average improvement in latency of processing data records for all applications by 48% in average compared to the round robin policy.

Future work. We do not consider neither auto-scaling nor migration issues in this project. We realized that the proposed controller has a certain upper bound on achieving its performance when running on a local cluster, particularly when a majority of computing modules suddenly receives a vast amount of incoming data elements. In such cases, the proposed controller needs to be equipped with a migration technique so that it launches more working threads and migrates some computing modules to newly launched threads to avoid QoS violation. Furthermore, we have only compared the controller with round robin algorithm as a classic simple algorithm. The experimental result of this report was quite simple, too, as it only includes a single set of experiments where all examined applications had the same structure, consisting of a single loop, So, the next step can be a comprehensive report for comparing the effectiveness and efficiency of the proposed method with some advanced sophisticated scheduling algorithms, such as (Lai, Fan, Zhang, and Liu 2015), to be validated more extensively under complex experimental settings.

ACKNOWLEDGEMENT

We would like to acknowledge the support by Australian Research Council (ARC) for the work carried out in this paper, under Linkage project scheme (LP160100406): *Cloud Resource Allocation under Bursty Conditions for Heavy-Tailed Job*. Samee U. Khan's work is supported by (while serving at) the National Science Foundation. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Abadi, M., F. McSherry, D. G. Murray, and T. L. Rodeheffer. 2013. "Formal analysis of a distributed algorithm for tracking progress". In *Formal Techniques for Distributed Systems*, pp. 5–19. Springer.

Abdelwahed, S. et al. 2009. "On the application of MPC techniques for adaptive performance management of computing systems". *IEEE Trans. on Network & Service Management* vol. 6 (4).

Apache Software Foundation 2016. "Storm, an open source distributed real-time computation system".

Ballani, H., P. Costa, T. Karagiannis, and A. Rowstron. 2011. "Towards predictable datacenter networks". In *SIGCOMM Computer Communication Review*, Volume 41, pp. 242–253. ACM.

Blumofe, R. D., C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. 1996. "Cilk: An efficient multithreaded runtime system". *Journal of parallel and distributed computing* vol. 37 (1), pp. 55–69.

Casalicchio, E., D. A. Menascé, and A. Aldhalaan. 2013. "Autonomic resource provisioning in cloud systems with availability goals". In *Cloud & Autonomic Computing Conf.*, pp. 1. ACM.

Chaiken, R., B. Jenkins et al. 2008. "SCOPE: easy & efficient parallel processing of massive data sets". *Proc. of VLDB Endowment* vol. 1 (2), pp. 1265–1276.

Chandu Thekkath 2017. "Naiad project".

Clauss, P.-N., and J. Gustedt. 2010. "Iterative computations with ordered read–write locks". *J. of Parallel & Distr. Computing* vol. 70 (5), pp. 496–504.

Dagum, P., R. Karp, M. Luby, and S. Ross. 2000. "An optimal algorithm for Monte Carlo estimation". *SIAM Journal on computing* vol. 29 (5), pp. 1484–1496.

De Matteis, T., and G. Mencagli. 2016. "Keep calm & react with foresight: Strategies for low-latency & energy-eff. elastic data stream proc.". In *Principles & Practice of Parallel Programming*, pp. 13.

Dean, J., and S. Ghemawat. 2008. "MapReduce: simplified data processing on large clusters". *Communications of the ACM* vol. 51 (1), pp. 107–113.

DeMatteis, T., and G. Mencagli. 2017. "Proactive elasticity and energy awareness in data stream processing". *J. of Systems & Software* vol. 127, pp. 302–319.

Dudoladov, S., C. Xu, S. Schelter, A. Katsifodimos, S. Ewen, K. Tzoumas, and V. Markl. 2015. "Optimistic recovery for iterative dataflows in action". In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 1439–1443. ACM.

Hellerstein, J. L., Y. Diao, S. Parekh, and D. M. Tilbury. 2004. *Feedback control of computing systems*. John Wiley & Sons.

HoseinyF., M., A. Y. Zomaya, and Z. Tari. 2017a. "A Model Predictive Controller for Managing QoS Enforcements and Microarchitecture-Level Interferences in a Lambda Platform". *IEEE Transactions on Parallel and Distributed Systems* vol. -.

HoseinyF., M., A. Y. Zomaya, and Z. Tari. 2017b. "QoS- & Contention-Aware Resource Provisioning in a Stream Proc. Engine". In *Cluster Computing (CLUSTER)*, pp. 137–146. IEEE.

Huang, X., G. Xue, R. Yu, and S. Leng. 2016. "Joint scheduling and beamforming coordination in cloud radio access networks with QoS guarantees". *IEEE Transactions on Vehicular Technology* vol. 65 (7), pp. 5449–5460.

Kleinrock, L. 1976. *Queueing systems, volume 2: Computer applications*, Volume 66. Wiley-Interscience New York.

Lai, P., R. Fan, W. Zhang, and F. Liu. 2015. "Utility Maximizing Thread Assignment and Resource Allocation". *arXiv preprint arXiv:1507.01101*.

Li, K., C. Liu, and K. Li. 2014. "An approximation algorithm based on game theory for scheduling simple linear deteriorating jobs". *Theoretical Computer Science* vol. 543, pp. 46–51.

Mencagli, G. 2016. "Adaptive model predictive control of autonomic distributed parallel computations with variable horizons and switching costs". vol. 28 (7), pp. 2187–2212.

Mencagli, G., M. Vanneschi, and E. Vespa. 2014. "A cooperative predictive control approach to improve the reconfiguration stability of adaptive distributed parallel applications". *ACM Trans. on Autonomous & Adaptive Systems (TAAS)* vol. 9 (1), pp. 2.

Microsoft Naiad 2017. "Microsoft Naiad: A Timely Dataflow System". https://github.com/MicrosoftResearch/Naiad. Accessed: 2017-11-6.

Murray, D. G., F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi. 2013. "Naiad: a timely dataflow system". In *Symposium on Operating Systems Principles*, pp. 439–455. ACM.

Neumeyer, L. et al. 2010. "S4: Distributed stream computing platform". In *Data Mining Workshops*, pp. 170–177. IEEE.

Padala, P. et al. 2009. "Automated Control of Multiple Virt. Resources". In *European Conf. on Computer Systems (EuroSys)*, pp. 13–26. NY, ACM.

Powell, W. B. 2007. *Approximate Dynamic Programming: Solving the curses of dimensionality*, Volume 703. John Wiley & Sons.

Şahin, S. 2015. *C-Stream: A coroutune-based elastic stream processing engine*. Ph. D. thesis, bilkent university.

Shen, Z., S. Subbiah et al. 2011. "Cloudscale: elastic resource scaling for multi-tenant cloud systems". In *Symposium on Cloud Computing*, pp. 5. ACM.

Wang, G. 2013. *Automatic Scaling Iterative Computations*. Ph. D. thesis, NY, USA.

Xiaoyong, T., K. Li, Z. Zeng, and B. Veeravalli. 2011. "A novel security-driven scheduling algorithm for precedence-constrained tasks in heterogeneous distributed systems". *IEEE Transactions on Computers* vol. 60 (7), pp. 1017–1029.

Yang, F., Z. Qian, X. Chen, I. Beschastnikh, L. Zhuang, L. Zhou, and J. Shen. 2012. "Sonora: A platform for continuous mobile-cloud computing". *Technical Report. Microsoft Research Asia, Tech. Rep*.

Zaharia, M., M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. 2010. "Spark: Cluster computing with working sets.". *HotCloud* vol. 10 (10-10), pp. 95.

Zou, Q., H. Wang et al. 2010. "From a stream of relational queries to distr. stream processing". *Proc. of the VLDB Endowment* vol. 3 (1-2), pp. 1394–1405.

AUTHOR BIOGRAPHIES

M.REZA HOSEINYFARAHABADY received the BSc degree in computer engineering and the MSc degree in information technology and network engineering both from Sharif University of Technology, Tehran, Iran, in 2005 and 2007, respectively. He received the PhD degree from the School of Information Technologies at the University of Sydney, in 2015. He is currently a research associate in the centre for Distributed and High Performance Computing, School of Information Technologies at the University of Sydney. His research interests include scheduling and resource allocation for parallel and distributed computing systems including Cloud computing and stream-lined dataflow applications, and control systems engineering. His email address is *reza.hoseiny@sydney.edu.au*.

SAEED BASTANI received his BSc degree (1998) in software engineering from Isfahan University of Technology, Isfahan, Iran. He received a MSc degree (2002) in artificial intelligence and robotics from Iran University of Science and Technology, Tehran, Iran, and his Ph.D. degree (2014) in computer science from University of Sydney, Australia. Prior to his Ph.D. studies, he worked for seven years as a researcher and developer in telecommunication sector. After completing his Ph.D. studies, he worked for six months as an associate researcher, focused on vehicular communication networks, at University of New South Wales, Australia. Since March 2014, he has been a researcher at Lund and Karlstad Universities in Sweden, where he has been involved in European projects in different roles, from research to leadership. His current research interests are performance modeling and simulation of networked systems including wireless networks, datacenter and cloud, Software Defined Networking (SDN), and Network Function Virtualization (NFV).

JAVID TAHERI received his BSc and MSc of Electrical Engineering from Sharif University of Technology, Tehran, Iran, in 1998 and 2000, respectively. He received his Ph.D. in the field of Mobile Computing from the School of Information Technologies at the University of Sydney, Australia. Since 2006, he has been actively working in several fields, including: networking, optimization, parallel/distributed computing, and cloud computing. He also holds several cloud/networking related industrial certification from VMware (VCP-DCV, VCP-DT, and VCP-Cloud), Cisco (CCNA), Microsoft, etc. He is currently working as an Associate Professor and a Docent at the Department of Computer Science in Karlstad University, Sweden. His major areas of interest are profiling, modelling and optimization techniques for cloud infrastructures, software defined networking and network function virtualization.

ALBERT Y. ZOMAYA is currently the Chair Professor of High Performance Computing and Networking and Australian Research Council Professorial Fellow in the School of Information Technologies, The University of Sydney. He is also the Director of the Centre for Distributed and High Performance Computing which was established in late

2009. He is the author/co-author of seven books, more than 500 publications in technical journals and conferences, and the editor of 14 books and 17 conference volumes. He is currently the Editor in Chief of the IEEE Transactions on Computers and serves as an associate editor for 20 journals including some of the leading journals in the field. Prof. Zomaya was the Chair the IEEE Technical Committee on Parallel Processing (1999-2003) and currently serves on its executive committee. He also serves on the advisory board of the IEEE Technical Committee on Scalable Computing, the advisory board of the Machine Intelligence Research Labs. Prof. Zomaya served as General and Program Chair for morethan60eventsand servedonthe committeesofmore than 600 ACM and IEEE conferences. He delivered more than 130 keynote addresses, invited seminars and media briefings. Prof. Zomaya is a Fellow of the IEEE, AAAS, the Institution of Engineering and Technology (U.K.), a Distinguished Engineer of the ACM and a Chartered Engineer (CEng). He received the 1997 Edgeworth David Medal from the Royal Society of New South Wales for outstanding contributions to Australian Science. He is also the recipient of the IEEE Computer Societys Meritorious Service Award and Golden Core Recognition in 2000 and 2006, respectively. Also, he received the IEEE TCPP Outstanding Service Award and the IEEE TCSC Medal for Excellence in Scalable Computing, both in 2011. His email address is *albert.zomaya@sydney.edu.au*.

ZAHIR TARI is a full professor in Distributed Systems at RMIT University (Australia). He received a bachelor degree in Mathematics from University of Algiers (USTHB, Algeria) in 1984, MSc in Operational Research from University of Greno- ble (France) in 1985 and PhD degree in Com- puter Science from University of Grenoble (France) in 1989. From 1990-1992, Zahir worked at the Database Laboratory at EPFL (Swiss Federal Institute of Technology) as a senior researcher, where he looked at various aspects of distributed database systems. In 1996, he moved to RMIT University as a senior lecturer and currently Professor, where he led the DSN (Distributed Systems and Networking) discipline. Zahirs expertise is in the areas of system performance (e.g., Web servers, P2P, Cloud) and system security (e.g., SCADA, Smart Grid, Cloud). He is the co-author of six books and he has edited over 25 conference proceedings. Zahir is a recipient of over 8M$ in funding from ARC (Australian Research Council) and lately part of a successful 7th Framework AU2EU (Australia to European) bid on Authorisation and Authentication for Entrusted Unions. Zahir was an associate editor of the IEEE Transactions on Computers (TC), IEEE Transactions on Parallel and Distributed Systems (TPDS) and IEEE Magazine on Cloud Computing. His email address is *zahir.tari@rmit.edu.au*.

SAMEE U. KHAN received a PhD in 2007 from the University of Texas, Arlington, TX, USA. Currently, he is a Program Director at the National Science Foundation, where he is responsible for the Smart & Autonomous Systems program, Critical Resilient Interdependent Infrastructure Systems and Processes program, and Computer Systems Research cluster. He also is a faculty at the North Dakota State University, Fargo, ND, USA. His research interests include optimization, robustness, and security of computer systems. His work has appeared in over 375 publications. He is on the editorial boards of leading journals, such as IEEE Access, IEEE Communications Surveys and Tutorials, IET Wireless Sensor Systems, Scalable Computing, IET Cyber-Physical Systems, and IEEE IT Pro. He is an ACM Distinguished Speaker, an IEEE Distinguished Lecturer, a Fellow of the Institution of Engineering and Technology (IET, formerly IEE), and a Fellow of the British Computer Society (BCS).