Cloud Resource Provisioning for Combined Stream and Batch Workflows

Raed Alsurdeh, Rodrigo N. Calheiros, Kenan M. Matawie and Bahman Javadi
School of Computing, Engineering and Mathematics
Western Sydney University
Sydney, Australia
Email: {r.aslurdeh, r.calheiros, k.matawie, b.javadi}@westernsydney.edu.au

***Abstract*—The increasing adoption of Internet of Thing (IoT) technology in many application domains generates a new need for rationalized utilization of computing resources supporting such computations. IoT applications can be represented as workflows in which stream and batch applications are integrated to accomplish data analytics objectives in many application domains such as smart home, health care, bioinformatics, astronomy, education, etc . The main challenge of this combination is the differentiation of service quality constraints between the two computation paradigms. Stream processing is highly sensitive to real-time constraint while batch processes are usually resource-intensive. In this work we propose a resource provisioning framework for combined workflows which aims to find an optimal workflow configuration plan to minimize execution time and monetary cost. The framework has functions of execution plan generation, task clustering, and resource provisioning. Results show that framework is capable to control the execution of combined-workflows by efficient tunning several parameters including stream arrival rate and processing throughput.**

I. Introduction

The increasing adoption of Internet of Thing (IoT) technology in many application domains generates a new need for rationalized utilization of computing resources to support such computations. The technology has been involved in many application domains such as smart home, health care, bioinformatics, astronomy, education, etc. By the year 2020, around 50 to 100 billion IoT-enabled devices will be connected to the Internet [1]. The upcoming popularity of IoT usage motivates researchers to introduce reliable and convenient application models to overcome the challenges of processing real-time data generated from IoT devices. These challenges are aligned with three main directions: integration and compatibility, security, and service quality. The majority of IoT-based applications are comprised of a large-scale and complex structure of interrelated tasks which are mostly high-resource demanding for computation, storage and bandwidth [2]. A common and efficient representation of such applications is Direct Acyclic Graph (DAG) structured workflow model. DAG representation that is widely accepted in many data analytics domains such as bioinformatics, high-energy physics, and astronomy.

Data analytics workflows have prominent features over traditional workflows that inspire researchers in proposing well-formed techniques and algorithms to optimize the increasingly cost of running large-scale versions of these workflows on commodity resources, such as public clouds in a stable and seamless manner. The structure of data analytics workflows embraces the integration of both stream and batch data processing pipelines. Stream and batch processing have different fundamental processing quality of service (QoS) requirements. IoT applications involve accepting high data stream rate from data channels and pipelines initiated from sources such as IoT devices and sensors.

As a first stage of data analytics workflow processing, data streams may need a real-time or near real-time preprocessing which may involve huge consumption of computation resources liable to stream format, processing time sensitivity, and nature of applied processing functions. On the other hand, batch processing which is more associated to functions of decision making and data aggregation over high data volume. Basically, batch processing is not a time-sensitive process, instead, it is more cost-effective due to the amount of resources required to handle large chunks of data. Moreover, both processing models are considerable for processing quality or throughput which determines the efficiency of processing computation under certain user and application QoS requirements.

To summarize, there are three main challenges associated with combined data analytics, namely, processing time sensitivity for stream data, monetary cost of resources, and throughput. So far and To best of our knowledge, none of the workflow resource estimation and scheduling techniques have advanced a detailed proposition of how to incorporate the structure of combined workflows (as a integration of stream and batch applications) and the above mentioned QoS processing challenges. This work aims to provide an adaptive resource estimation and provisioning framework for combined workflows. This paper has the following contributions:

- A resource provisioning algorithm for optimizing monetary cost and execution time for combined workflows considering the characteristics of stream and batch applications, and the dependencies between tasks.
- An adjustment-based resource estimation algorithm for stream applications using an evolutionary technique. The algorithm adjusts the stream arrival rate and aggregation windows size to optimize the cost of resources while satisfying the task execution throughput.
- A cluster-based resource provisioning strategy. The strategy implies grouping non-dependent tasks and perform a

cumulative resource provisioning in a periodic manner which can minimize the monetary cost for long-term workflow executions.

The rest of paper is structured as follows. In Section II, we discuss related work on workflow resource provisioning. Detailed description of application modeling, problem formulation, and the resource provisioning framework is provided in Section III and IV, respectively. Section V shows the experimental findings and provides insights on main results. Finally, the conclusion and future works are provided in Section VI.

II. RELATED WORK

Cloud resource provisioning has received high attention in recent years. Many models have been proposed to overcome the challenge of reducing resource usage cost while balancing other quality measurements [7], [14], [15], such as response time [3], scalability [4], energy [5], resource failure [6] and throughput. Basically, resource provisioning relies on two main criteria: application workload and complexity [7], and provisioning time sensitivity. Most of resource provisioning models assume deterministic workload models, which are high predictable, and mostly focus on generating resource allocation plans to reduce resource allocation cost. Iqbal et al. [8] observed that even adaptive resource provisioning do not scale workload automatically, but may do it per user signal. Kapitanova et al. [9] employed stream application execution time in predicting data workload in a fixed data size mode, and Zhang et al. [10] proposed an energy-aware dynamic resource provisioning model using a queuing system model. Their work does not consider the workload characteristics. Vecchiola et al. [3] presented a multi-cloud resource provisioning model to enhance scientific workflows QoS constraints while meeting execution deadline. Shao et la. [11] provided a provisioning model for big data applications, their workload model emphases only the number of jobs. Warneke and Kao [12] proposed *Nephele*, a cloud-based parallel processing framework, and characterized three features of sufficient resource provisioning model: understand the dependences between application tasks, analyze the specifications of available resources, and provide efficient resource allocation and deallocation model. Li et al. [13] concluded that it is not trivial to determine the characteristics of data stream application for the purpose of resource provisioning and allocation.

We can conclude from related work discussion the emphasize on quantitative analysis for incoming application workloads while provide provisioning techniques. Moreover, non of these models have stressed the fabric representations of these applications as many scientific and business domains are moving toward combined dataflows of stream and batch applications.

This work provides a resource provisioning framework for combined workflow applications to meet the mentioned research limitations as we will consider the dependencies between steam and batch applications, The framework will provide an optimized resource previsioning plan based on mentioned considerations.

III. APPLICATION MODEL

The work is proposed to estimate and provision cloud resources for combined workflows which have the integration of stream and batch tasks (applications). A workflow w is modeled as a Direct Acyclic Graph (DAG) of tasks T with direct edges E and no cycles or conditional dependencies, w = $G(T, E)$, where T is a set of tasks T = $\{t_1, t_2, \ldots, t_n\}$ and E is a set of edges in which an edge e_{ij} is exist if there is a data dependency between tasks t_i and t_j. A task t_i represents the execution of either a streaming (${t_i}^S$) or batch (${t_i}^B$) application. The two type of applications have different characteristics which formed the modeling process, specifically, a streaming task is modeled as a $M/G/c$ multi-server queuing system [16], where a batch task is modeled as a simple deterministic mathematical model. We used the term task and application interchangeably in this work, and we assume that a task itself is a composition of interrelated sub-tasks. The work does not handle the inner-representation of tasks and keeps the abstraction at application-level only.

A. Stream Application Model

We assumes that each stream task ${t_i}^S$ represents the execution of an end-to-end stream application. For the purpose of resource estimation, we modeled stream task execution ${t_i}^S$ as an $M/G/c$ queuing system [16], by which the number of servers is calculated. A stream task ${t_i}^S$ is modeled as:

$$ {t_i}^S = \{\lambda_i, \mu_i, \omega_i, \tau_i, \alpha_i\} \tag{1} $$

where λ_i is the application stream arrival rate (msg/s), μ_i is the server service rate (msg/s), ω_i is the stream processing aggregation window size (s), τ_i is the application minimum throughput, we defined the throughput as the percentage of processed stream messages in a given aggregation window, and α_i is application data production factor which determines the amount of data to be passed to the next batch task(s).

B. Queuing system model for stream applications

The main objective of the adopted queuing modeling is to estimate the number of servers needed to run the application under constraints of system utilization , waiting time, and system throughput. Number of servers is related to the parallelization level of a stream application, which is an effective technique for seamless and transparent application auto-scaling [19].

The adopted queuing system has the following aspects: a data stream arrives to an infinite waiting queue that is served by c identical servers, the server service time is a random variable with general distribution and mean $1/\mu$, the interarrival times between element arrivals to the queue is random variable with mean $1/\lambda$, and all service and interarrival times are assumed independent. Arriving stream elements are served in First-come First-served manner (FCFS) and only one stream element may receive a service from a server at a given time. Such system is often referred to as $M/G/c$, following Kendalls notation [17] (M indicates Poisson distribution for interarrival rate and G indicate general distribution for service times, and c is the number of servers).

Modeling a stream application as a $M/G/c$-queue helps in predicting the performance measurements of an application, such as resource utilization and data throughput. Dor et al. [18] investigated the correspondence between a relatively simple queuing model of an FPGA-accelerated BLAST implementation and empirical measurements taken from executions of the actual application. The study shows that simple queuing networks can accurately model the performance of a heterogeneous streaming application. Li et al. [13] argue that queuing system has a better performance under certain assumptions of a data stream application.

We assume that at a given moment of time, all servers c will be busy. Based on that, Hokstad [20] proved that we can treat $M/G/c$ with service time ($S = 1/\mu$) as a $M/G/1$ with service time ($S = 1/\mu c$). We will use the approximation by Kleinrock [16] as:

$$\rho = \frac{\lambda}{\mu c} \tag{2}$$

$$W_q = \frac{\rho(1 + S^2)}{2(1 - \rho)\mu} \tag{3}$$

$$ES^2 = \sigma_s^2 \mu^2 \tag{4}$$

where:

ρ	System utilization, assume to be < 1
W_q	Waiting time in queue
σ_s^2	The variance of service time
S^2	The coefficient square of the service time variance

C. Batch Application Model

Similar to stream task, we assumes that each batch task ${t_i}^B$ represents the execution of an end-to-end batch application. In order to estimate the number of servers, we modeled ${t_i}^B$ as a simple deterministic mathematical model:

$${t_i}^B = \{\vartheta_i, \tau_i, \alpha_i, \beta_i\} \tag{5}$$

where ϑ_i is the application deadline (s) , τ_i is the application minimum throughput , α_i is the application data production factor, and β_i is the application aggregation factor which denotes to the number of messages required for producing a single application output.

IV. Resource Estimation and Provisioning Framework

This section discusses the proposed resource estimation and provisioning framework for combined workflows. Fig 1 shows the main framework components and dependencies between these components, which are explained in the following:

1) Queueing System Builder: The first step after receiving user workflow is constructing and validating queueing systems for stream tasks. This step validates workflow liability as a framework input.

2) Workflow Configuration Plan Generator: This is the key component of the framework in which workflow configuration plans are generated under constraints of queueing system utilization, waiting time and workflow execution throughput. A configuration plan is a set of values for stream applications refer to properties of window size and arrival rate. Based on the dependencies between workflow tasks, these values have direct influence on measuring execution time and monetary cost of resources for both stream and batch applications. For the purpose of generating configuration plans, we adopted Particle Swarm Optimization (PSO) technique [21].

PSO is a stochastic global optimization method introduced by Eberhart and Kennedy [21] which based on simulation of social behavior. As in Genetic Algorithm (GA) and Evaluation Strategies (ES), PSP exploits a population of potential solutions to prop the search space. PSO relies on exchange information between individuals, called *particles*, of the population, called *swarm*. Each particle adjusts its trajectory towards its own previous best position (*local best*), and towards the best previous position attained by the entire population (*global best*). This behavior improves the converge time to get a global minima with a reasonable good solution. A particle movement (new position) is co-ordinated by the velocity which has both magnitude and direction. The particle velocity is influenced by the particle best position and the global best position, and also controlled by parameters including inertia weight and acceleration coefficients.

The objective of adopting PSO is to generate adjusted workflow configuration plans by randomizing data stream arrival rate λ and aggregation window size ω in order to minimize the optimization objective function and meeting the constraints of throughput τ and deadline ϑ for each streaming and batch application, respectively. This kind of PSO modeling is called constrained-PSO optimization [22]. Fitting constrained-PSO method to the proposed optimization problem will identify both the fitness function and the particle structure. The fitness

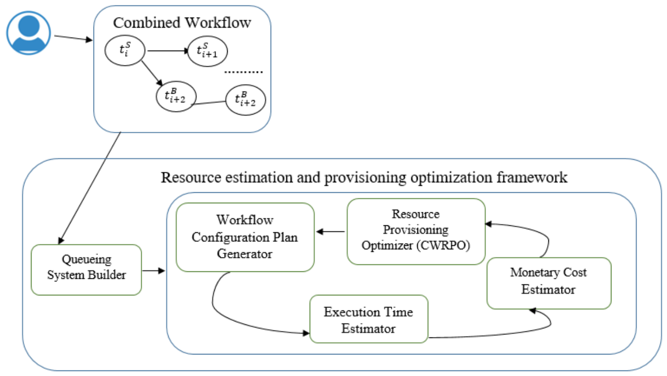
Combined Workflow
t_i^S
t_{i+1}^S
t_{i+2}^B
t_{i+2}^B
Resource estimation and provisioning optimization framework
Queueing System Builder
Workflow Configuration Plan Generator
Resource Provisioning Optimizer (CWRPO)
Monetary Cost Estimator
Execution Time Estimator

Fig. 1. Resource estimation and provisioning framework

Algorithm 1 Finding an optimal workflow configuration plan

```
Load PSO Configuration
Initialize Particles $P$
$P.pos \leftarrow NULL$                                ▷ Best Position
$P.gbest \leftarrow \inf$                              ▷ Best Cost
for $i \leftarrow 1, n$ do                             ▷ $n$: Number of particles
    Randomize $P_i.pos$
    $P_i.velcity \leftarrow 0$
    $P_i.cost \leftarrow callculteCost(P_i.pos)$       ▷ Algorithm. 5
    $P_i.lbest \leftarrow P_i.cost$
end for
for $j \leftarrow 1, m$ do                             ▷ $m$: Number of iterations
    for $i \leftarrow 1, n$ do
        Update $P_i.velcity$
        $P_i.pos \leftarrow P_i.pos + P_i.velcity$      ▷ move particle
        $P_i.cost \leftarrow callculteCost(P_i.pos)$
        if $P_i.cost \leq P_i.lcost$ then
            $P_i.lbest \leftarrow P_i.cost$            ▷ update local best
            if $P_i.lbest \leq P.gbest$ then
                $P.gbest \leftarrow P_i.lbest$         ▷ update global best
                $P.pos \leftarrow P_i.pos$
            end if
        end if
    end for
end for
if $P.gbest == \inf$ then
    return NULL                                        ▷ No solution
else
    return $P.pos$
end if
```

function, which also called the objective function, measures the performance of a particle for comparison purpose with local and global optimum. The optimization function expresses the fitness function. To serve the objective PSO engine for adjusting the values of arrival rate λ and window size ω, a particle is structured to hold required values for all stream tasks, and the particle dimension equals the number of stream tasks. Thus, the particle structure is a workflow-dependent where it's complexity derived from the number of stream tasks as well as the workflow structure complexity. For example, Table I shows how a 5-dimensional particle which is structured for a workflow with 5-stream applications. Columns of Arrival rate λ and Window size ω values express the particle position in a random iteration. Moreover, a particle position is the representation of a workflow configuration plan which will feed the cost computation process to generate values which will be applied for the objective function.

Algorithm 1 provides the steps to find the optimal configuration plan for a given combined workflow. The PSO-based technique has two main steps: generates solutions (configuration plans), and evaluates the performance of these solutions. The first step is handled by providing randomizations for particles and moving particles toward calculated velocity. Algorithm 2 shows the computation procedure to calculate the optimization

Algorithm 2 Tasks Clustering (Periods) and Critical Path

```
procedure FINDWORFLOWCP($T$)
    Find Workflow Critical Path $CP$
    ETP $\leftarrow \{\}$                               ▷ Workflow Execution Periods
    for $i \leftarrow 1, n$ do                          ▷ $n$: Number of tasks
        if $T_i \in CP$ then
            Add $T_i$ to $ETP$
        else
            Try to allocate $T_i$ in $ETP$
        end if
        Add $T_i.S$ to $ETP.S$
                                     ▷ Update period number of servers
    end for
    if All Tasks allocated then
        return $CP$ , $ETP$
    else
        return NULL
    end if
end procedure
```

value of a configuration plan, which is represented as a PSO particle position. To evaluate the performance for a given solution, we need to calculate end-to-end workflow execution time and total resource provisioning monetary cost. .

3) Execution Time Estimator: The workflow execution time is estimated using task clustering approach. Task clustering is the process of grouping independent tasks in time periods. Each period of time has the length of leading task in workflow execution critical path (CP). CP represents the longest-time to process the workflow, and critical tasks the most significant contribution on workflow execution time. We consider that each critical task as a separate period of execution time, and then try to converge all remaining tasks to appropriate periods. Algorithm 2 provides a simple conceptualization of task clustering. However, this concept allows simplification of resource provisioning as it eliminates the complexity of allocating resources for interrelated tasks which will be in different periods of time. Finally, period-based resource provisioning. additional dimension of cost reduction is selecting a cost-effective resource allocation plan in respect to the variety of cloud computation units (virtual machine). On the top of that,

4) Monetary Cost Estimator: We assumed resource provisioning at group-basis, it means allocating resources for all tasks included in a period of time as Bag of Task (BoT)

TABLE I
AN EXAMPLE OF PARTICLE POSITION (A CONFIGURATION PLAN)

Task Index i	**Arrival Rate** (λ') **(msg/s)**	**Window Size** (ω') **(s)**
2	4792	275
3	4250	344
7	3989	250
8	3685	1100
12	3700	1250

Algorithm 3 Periodic Resource provisioning

```
1:  procedure PROVIOSINPERIODRESOURCES($ETP_i$)
2:      load $VM$                    ▷ Get cloud resource configuration
3:      C˙min ← inf                  ▷ Minimum $ETP$ Cost
4:      for $i \leftarrow 1, v$ do
5:                                   ▷ $v$: Number of cloud $VM$ configurations
6:          $C_{VMi} \leftarrow provisionETP(ETP.S, VM_i)$
7:          if $C_{VMi} \leq C_{min}$ then
8:              $C_{min} \leftarrow C_{VMi}$
9:          end if
10:     end for
11:     return $C_{min}$
12: end procedure
```

resource provisioning problem. Rodriguez and Buyya [23] demonstrated that fine-grained resource provisioning for BoT applications can reduce the execution time while achieving budget constraint. Periodic provisioning technique is adopted, which has two steps: finding number of servers, and finding the optimized resource provisioning plan. For a stream task, the process is straightforward, values are constructed from the proposed queuing system. Furthermore, number of servers c_s represents the level of parallelism. We assume that each level of parallelism equivalent to a single-core machine. from Equation 2, we can derived number of servers c_s as :

$$c_s = \frac{\lambda'}{\mu\rho} \tag{6}$$

For a batch task, Algorithm 4 shows execution time and cost calculations upon incoming computation load. After finding the total number of servers (equivalent to number machine cores) for a period of time, a deterministic model is used to find optimized (less cost) provisioning plan on the available computation machines (VMs). Algorithm 5 discuses the logic of finding the resource provisioning plan. Periodic provisioning technique is described in Algorithm 3.

5) Combined-Workflow Resource Provisioning Optimizer (CWRPO): The optimizer aims to control the execution of combined workflows in order to minimize monetary cost and execution time. In other words, reducing provisioned resources cost C while minimizing the end-to-end execution time E for each batch application ${t_i}^B$. A configuration plan is evaluated based on the following optimization function:

$$min(C * E) \tag{7}$$

Subject to:

$$E_{t_i} < \vartheta_{{t_i}^B} \ , \forall {t_i}^B \in T$$
$$\tau_{{t_i}^S} > \tau_{t_D} \ , \forall {t_i}^S \in T$$

Where:

E_{t_i}	Batch Task execution time
ϑ_{t_i}	Task deadline
τ_{t_i}	Stream Task throughput
τ_{t_D}	Application-defined throughput

Algorithm 4 Process batch task T_i to find number of servers and execution time

```
procedure PROCESSBATCHTASK($T_i$)
    $S \leftarrow 0$                         ▷ Number of servers
    $ET \leftarrow 0$                        ▷ Execution Time
    Compute total predecessor stream tasks load $totalSLoad$
    Compute total predecessor batch tasks load $totalBLoad$
    $totalLoad \leftarrow totalSLoad + totalBLoad$
    Compute total load time $totalBTime$
    while $totalBTime \geq \vartheta$ do     ▷ Check deadline
        $S \leftarrow S + 1$
        $updateET$
    end while
    return $S, ET$
end procedure
```

Algorithm 5 Calculating the optimization value of a configuration plan

```
1:  procedure CALLCULTECOST($Pos$ , $T$)
2:      Parameters: Position $Pos$ , Workflow Tasks $T$
3:      $C \leftarrow 0$                         ▷ Monetary Cost
4:      $ET \leftarrow 0$                        ▷ Execution Time
5:      for $i \leftarrow 1, n$ do              ▷ $n$: Number of tasks
6:          if $T_i.type == STREAM$ then
7:              $T_i.S \leftarrow findStreamS(P_i.\lambda)$        ▷ Eq. 6
8:              $T_i.ET \leftarrow P_i.\omega$          ▷ $ET$ is the window size
9:          else
10:             $T_i.S, T_i.ET \leftarrow ProcessBatchTask(T_i)$
11:                                                  ▷ Algorithm. 4
12:         end if
13:     end for
14:     $ET, ETP \leftarrow findWorflowCP(T)$    ▷ Algorithm. 2
15:     if $ET \neq NULL$ then
16:         for $j \leftarrow 1, p$ do ▷ $p$: Number of execution periods
17:             $C \leftarrow C + proviosinPeriodResources(ETP_j)$
18:                                                  ▷ Algorithm. 3
19:         end for
20:         return $C * ET$
21:     else
22:         return inf
23:     end if
24: end procedure
```

V. Performance evaluation

In this section, we present the experiment setup and results of the proposed framework.

A. Experimental setup

Shukla et al. [24] provide valuable benchmarking results for standard IoT application dataflows for both stream and batch applications. In addition, they discussed the standard structure of these application by identifying the dependencies

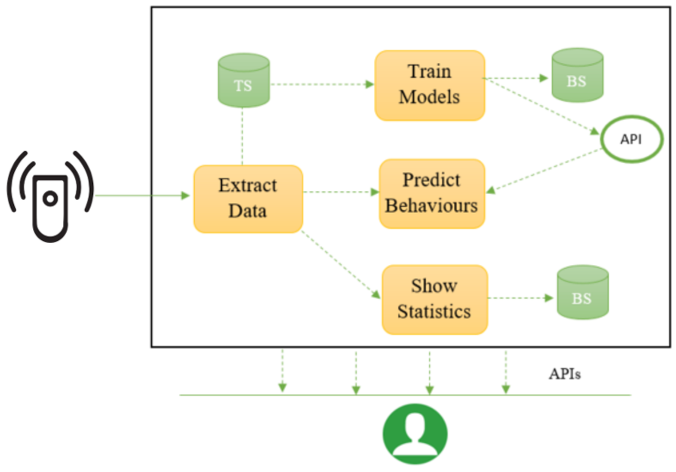

Fig. 2. High level stream and batch applications dependencies for data analytics workflows [24]

between stream and batch tasks. Fig 2 shows four IoT-based dataflow application tasks. Extract data from steams and perform predictions are stream tasks, whereas training machine learning models and showing statistical results are more related to batch processing. Shukla et al. [24] evaluated each application type based the micro-benchmarking for sub-tasks, and by running each sub-task in a single-core machine. We used the benchmarking results in defining initial models parameters as well as building workflows for experiments. Three workflows were constructed: small, medium horizontal-scale, and large vertical-scale. Workflow scalability has a high correlation with optimization parameters. Based on our framework, workflow execution time should be more sensitive to vertical scalability which horizontal scalability has more contribution in the total monetary cost. Table II presents the main characteristics of these workflows.

We extended CloudSim [25] to support the execution of workflows. We modeled a single data center and three VM types to simulate a cloud resource provider. The VM types have the same configurations of Amazon EC2 VMs. Table III shows the details about VM configurations.

We compare the performance of the proposed CWRPO with two other techniques, namely, full mode and random selection. Full mode technique implies running the workflow with semi-optimization process at queuing system utilization level. Random selection technique is a provisioning technique without optimization efforts at any framework execution level, either on queuing system utilization or task clustering.

TABLE II
WORKFLOWS CHARACTERISTICS

Workflow	#Stream Tasks	#Batch Tasks	Scale Mode
Small	3	5	Equal
Medium	15	15	Horizontal
Large	25	42	vertical

TABLE III
TYPES OF VMS USED IN PERFORMANCE EVALUATION

Name	CPU capacity (MIPS)	Price per hour
t2.large	2	\$0.0928
t2.xlarge	4	\$0.1856
t2.2xlarge	8	\$0.3712

B. *Results and discussions*

This section discuses and investigates the results of applying the resource estimation and provisioning framework on three different combined workflows. We evaluated the association between the variation in model parameters (window size, arrival rate, and throughput) and workflow scalability (vertical and horizontal), and optimization parameters (execution time and monetary cost). For each model parameter, simulation is carried out 30 times, and average values are used for comparing the performance of Full Mode, CWRPO, and Random Selection techniques. We varied model parameter values in range of 25% to 175%. Relative percentages have been proposed due to the variation in parameter values among workflow tasks.

1) Window size: is the time length of processing and aggregating incoming data stream. Figures 3-5 show the results of varying window size on execution time and monetary cost. CWRPO demonstrates high stability in controlling the workflow execution time with increase in window size length. In Fig 4 and 5, CWRPO shows slight difference execution time optimization with horizontal scalability over vertical scalability. With horizontal scalability, CWRPO was able to produce cost reduction linearly with increase in window size with about 80% reduction compared to Full-Mode technique. This is refers to ability to cluster/group higher number tasks/applications and performs periodic resource previsioning in an effective manner comparing to vertical scalability with 30% maximum cost reduction. This allows IoT applications to effectively produce more data either by adding more IoT devices (horizontal scaling) or dividing the incoming data load on applications (vertical scaling).

2) Arrival Rate: Results from Fig 6-8 show that CWRPO can have steady control on the incoming arrival rate under the constraints of queueing system and throughput. For all simulated workflow structures, CWRPO demonstrated execution time reduction compared with other techniques. However, Fig 8 show less efficiency in execution reduction with 14% on average, compared to 25% from Fig 7. This can be explained by recognizing the high efficiency in cost reduction with average 60% and 40% in case of vertical and horizontal scalability, respectively. In fact the impact of arrival rate on monetary cost is strongly related with throughput constraint. As the constraint is relaxed, the algorithm will try to drop as many as possible of incoming stream messages. This scenario is convenient for IoT application models where performance is not aligned to generated data volume. For instance, modeling of rare phenomena prediction such as Earthquakes and floods.

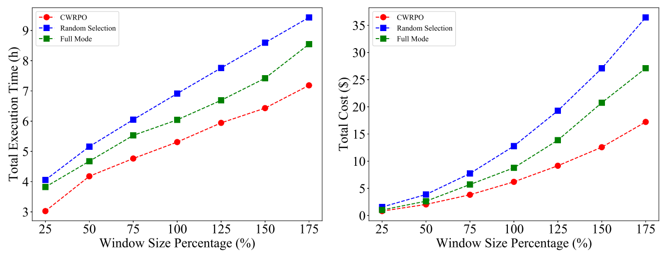

Fig. 3. Window size variation impact on the Small workflow

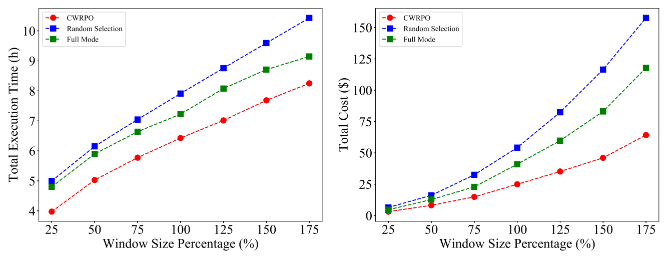

Fig. 4. Window size variation impact on the Medium workflow

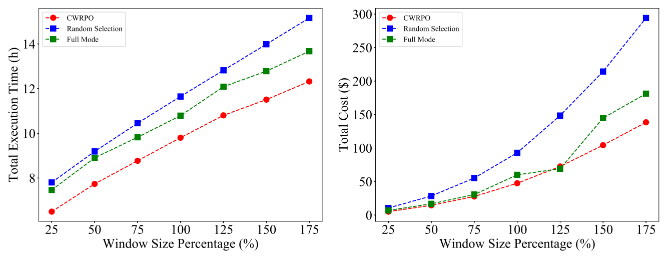

Fig. 5. Window size variation impact on the Large workflow

3) Throughput: In case of throughput, we varied the constraint in range 40-70%. Fig 9-11 show that monetary cost is exponentially increased while throughput is moving to it's peak value. In complex workflows, Fig 10 and 11, the increase in throughput value drives CWRPO to add more resources to allow processing more data. For example, in vertical scaling example, Fig 11, a 400% additional cost is required when moving from 40% to 70% throughput. Setting-up the throughput constraint is an application and performance dependent, in which, advanced heuristics algorithms and tunning techniques can produce significant cost reduction with complex and long-term combined workflow execution.

VI. Conclusion and Future Work

In this work, we presented a resource estimation and provisioning framework, CWRPO, for combined workflows which are integration of stream and batch applications. The framework aims to generate optimized workflow configuration plan in which the workflow execution time and monetary cost can be reduced. We developed a simulation environment to evaluate the the influence of resource provisioning model

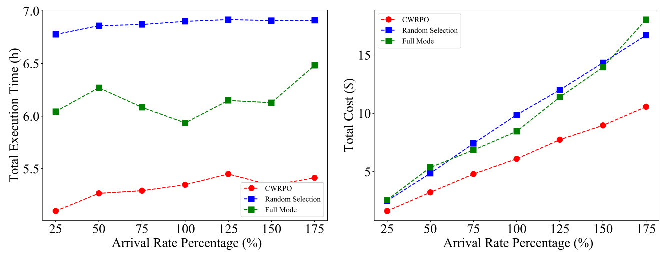

Fig. 6. Arrival rate variation impact on the Small workflow

parameters, namely, arrival rate, window size and throughput, on the optimization objective. In addition, we applied the framework on different workflow structures with various scalability factor.

We compared our technique (CWRPO) with baseline (Random Selection) and Full Mode techniques. Overall, CWRPO demonstrates promising execution time and cost reduction in most of parameter variations for the three workflow cases. Apart from other parameters, window size has the obvious

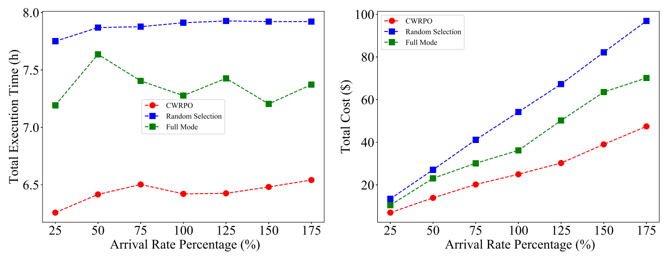

Fig. 7. Arrival rate variation impact on the Medium workflow

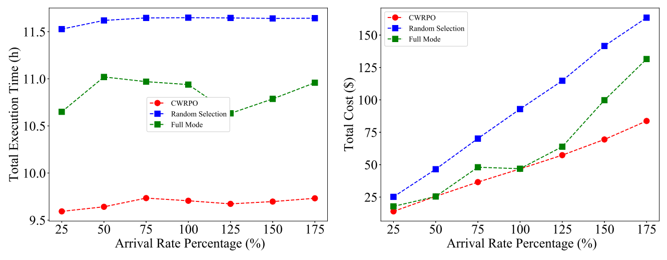

Fig. 8. Arrival rate variation impact on the Large workflow

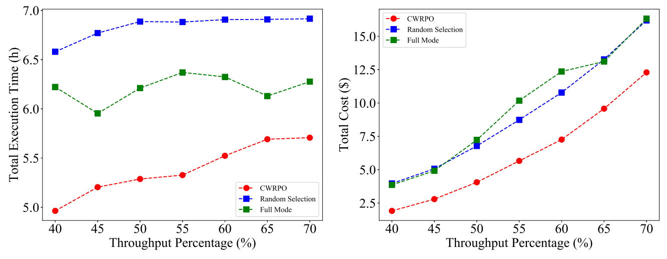

Fig. 9. Throughput variation impact on the Small workflow

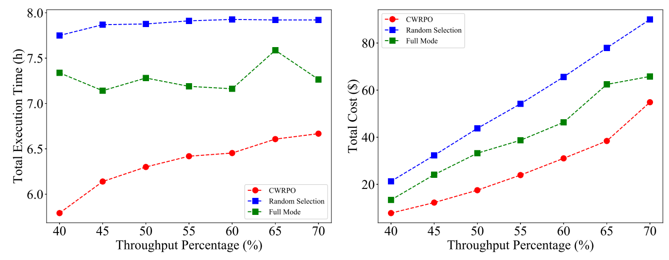

Fig. 10. Throughput variation impact on the Medium workflow

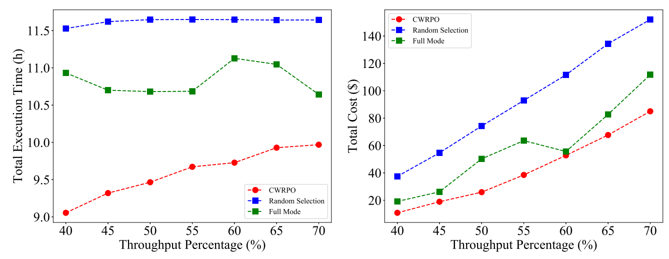

Fig. 11. Throughput variation impact on the Large workflow

influence on both execution time and cost. Results showed that CWRPO was able to control through the efficient adoption of task clustering and periodic resource provisioning techniques. In addition, results showed the correlation between arrival rate and throughput. IoT-based workflows can have efficient cost-reduction with minimized throughput constraint. Furthermore, the analysis highlighted the sensitivity of optimization objective to throughput constraint, and the necessity of building efficient tunning techniques to guarantee reasonable margin of workflow execution optimization.

In the future, we are planning to conduct more experiments on more complex combined workflows with different variations of data workloads. We also planning to extend the work by adding components of workflow scheduling and model parameters tunning.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] S. Sagiroglu and D. Sinanc, "Big data: A review," in *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, pp. 42–47, IEEE, 2013.

[3] C. Vecchiola, R. N. Calheiros, D. Karunamoorthy, and R. Buyya, "Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 58–65, 2012.

[4] R. Chard, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, and I. Foster, "Cost-aware elastic cloud provisioning for scientific workloads," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pp. 971–974, IEEE, 2015.

[5] Y. Sharma, B. Javadi, W. Si, and D. Sun, "Reliable and energy efficient resource provisioning and allocation in cloud computing," in *Proceedings of the10th International Conference on Utility and Cloud Computing*, pp. 57–66, ACM, 2017.

[6] B. Javadi, J. Abawajy, and R. Buyya, "Failure-aware resource provisioning for hybrid cloud infrastructure," *Journal of parallel and distributed computing*, vol. 72, no. 10, pp. 1318–1331, 2012.

[7] S. Singh and I. Chana, "Cloud resource provisioning: survey, status and future research directions," *Knowledge and Information Systems*, vol. 49, no. 3, pp. 1005–1069, 2016.

[8] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 871–879, 2011.

[9] K. Kapitanova, S. H. Son, W. Kang, and W.-T. Kim, "Modeling and analyzing real-time data streams," in *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2011 14th IEEE International Symposium on*, pp. 91–98, IEEE, 2011.

[10] Q. Zhang, M. F. Zhani, S. Zhang, Q. Zhu, R. Boutaba, and J. L. Hellerstein, "Dynamic energy-aware capacity provisioning for cloud computing environments," in *Proceedings of the 9th international conference on Autonomic computing*, pp. 145–154, ACM, 2012.

[11] Y. Shao, C. Li, J. Gu, J. Zhang, and Y. Luo, "Efficient jobs scheduling approach for big data applications," *Computers & Industrial Engineering*, vol. 117, pp. 249–261, 2018.

[12] D. Warneke and O. Kao, "Exploiting dynamic resource allocation for efficient parallel data processing in the cloud," *IEEE transactions on parallel and distributed systems*, vol. 22, no. 6, pp. 985–997, 2011.

[13] T. Li, J. Tang, and J. Xu, "A predictive scheduling framework for fast and distributed stream data processing," in *Big Data (Big Data), 2015 IEEE International Conference on*, pp. 333–338, IEEE, 2015.

[14] S. S. Manvi and G. K. Shyam, "Resource management for infrastructure as a service (iaas) in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 41, pp. 424–440, 2014.

[15] A. Hameed, A. Khoshkbarforoushha, R. Ranjan, P. P. Jayaraman, J. Kolodziej, P. Balaji, S. Zeadally, Q. M. Malluhi, N. Tziritas, A. Vishnu, *et al.*, "A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems," *Computing*, vol. 98, no. 7, pp. 751–774, 2016.

[16] L. Kleinrock, *Queueing systems, volume 2: Computer applications*, vol. 66. wiley New York, 1976.

[17] D. G. Kendall, "Some problems in the theory of queues," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 151–185, 1951.

[18] R. Dor, J. M. Lancaster, M. A. Franklin, J. Buhler, and R. D. Chamberlain, "Using queuing theory to model streaming applications," in *Symp. on Application Accelerators in High Perf. Computing*, 2010.

[19] S. Schneider, M. Hirzel, B. Gedik, and K.-L. Wu, "Auto-parallelizing stateful distributed streaming applications," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pp. 53–64, ACM, 2012.

[20] P. Hokstad, "On the steady-state solution of the m/g/2 queue," *Advances in Applied Probability*, vol. 11, no. 1, pp. 240–255, 1979.

[21] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pp. 39–43, IEEE, 1995.

[22] K. E. Parsopoulos, M. N. Vrahatis, *et al.*, "Particle swarm optimization method for constrained optimization problems," *Intelligent Technologies–Theory and Application: New Trends in Intelligent Technologies*, vol. 76, no. 1, pp. 214–220, 2002.

[23] M. A. Rodriguez and R. Buyya, "Budget-driven resource provisioning and scheduling of scientific workflow in iaas clouds with fine-grained billing periods," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 9, no. 4, 2015.

[24] A. Shukla, S. Chaturvedi, and Y. Simmhan, "Riotbench: An iot benchmark for distributed stream processing systems," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 21, p. e4257, 2017.

[25] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.