BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **22**, No 3
Sofia • 2022 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2022-0027

Uncertainty Aware T2SS Based Dyna-Q-Learning Framework for Task Scheduling in Grid Computing

K. Bhargavi[1], *Sajjan G. Shiva*[2]

[1]*Department of CSE, Siddaganga Institute of Technology, Tumakuru, Karnataka, India*
[2]*Department of CS, University of Memphis, Memphis, Tennessee, USA*
E-mails: bhargavik@sit.ac.in Sshiva@memphis.edu

Abstract**:** *Task scheduling is an important activity in parallel and distributed computing environment like grid because the performance depends on it. Task scheduling gets affected by behavioral and primary uncertainties. Behavioral uncertainty arises due to variability in the workload characteristics, size of data and dynamic partitioning of applications. Primary uncertainty arises due to variability in data handling capabilities, processor context switching and interplay between the computation intensive applications. In this paper behavioral uncertainty and primary uncertainty with respect to tasks and resources parameters are managed using Type-2-Soft-Set* (*T2SS*) *theory. Dyna-Q-Learning task scheduling technique is designed over the uncertainty free tasks and resource parameters. The results obtained are further validated through simulation using GridSim simulator. The performance is good based on metrics such as learning rate, accuracy, execution time and resource utilization rate.*

Keywords**:** *Dyna-Q-Learning, grid computing, T2SS, uncertainty, performance.*

1. Introduction

In recent years there is a tremendous development of grid computing environment, which supports applications that have high computation demands. Compared to other High-Performance-Computing (HPC) environments like cluster computing and cloud computing, the grid computing allows large scale heterogeneous computation to be performed at speed among the geographically distributed nodes [1, 2]. Although individual grids are customized to perform specific operations, but they are even extended to perform variety of other operations multimodal approximations, simultaneous analysis of data, large scale sharing of grid resources, belief propagation, approximate factorization, topic modelling, distributed arithmetic, high resolution image indexing, logic synthesis, etc. However grids are subjected to lot of performance issues due to uncertainty in the computing environment that include inadequate excitation, poor signal to noise ratio for input data, progressive refinement of grids, higher number of grid operators, discretization of uncertainty measurement, low fidelity model outputs, randomness of input parameters, dispersion of resources, vulnerability of problem, complexity of unstructured data, curse of computational dimensionality, suboptimal task scheduling, poor fault tolerance, cascading grid failures, and many more. Hence, there is a necessity to manage uncertainty in the input and output parameters to improve endurance performance for longer duration of time [3-5].

Type 2 Soft Set (T2SS) is one of the promising types of mathematical framework which oversees the parametric uncertainty (vagueness) by drawing soft boundary over the parameters. It is being applied in various applications that include function smoothness, integration, game theory, scientific computing, internet computing, signal processing, data science, control theory, thermal analysis, robotics, data intensive computation, machine control, load unbalancing, etc. [6, 7].

Dyna-Q-Learning is a kind of reinforcement learning technique which formulates policies from both real experiences and simulation experience. The technique works in three phases planning, acting, and learning in which value function (policy function) gets improved through direct learning or indirect learning of the agent. For planning and direct reinforcement learning one-step tabular Q-Learning is employed and whenever the model is queried the last observed state and the reward are returned as prediction. Both planning and learning phases of Dyna-Q-Learning agent learn from various sources of information through common final path [8].

Task scheduling is an important activity in HPC domain, the main objective of the task scheduler is to improve the system performance by satisfying the resource requirement of the tasks by efficiently managing uncertainty. In this paper the parametric uncertainty of grid environment is overseen by performing double approximation of the parameters using T2SS mathematical framework. Better task scheduling policies are formulated with few interactions with the environment through efficient utilization of the direct and indirect experiences of the Dyna-Q-Learning agent [9, 10].

The main objectives of the paper are as follows.

- Mathematical representation of the grid computing system model for task scheduling by setting up definitions for the performance metrics.
- Identifying the uncertainty in the grid resource and task parameters using Hidden Markov Model (HMM) and Partially Observable Markov Decision Process (POMDP) model.
- Design of a novel T2SS based Dyna-Q-Learning task scheduling framework supported by algorithms for each of the component of the framework.
- Expected value analysis of the proposed Dyna-Q-Learning task scheduling under finite and infinite computing scenarios.
- Simulation of the Dyna-Q-Learning task scheduler using GridSim simulator by considering three different type of workloads that are MetaCentrum, Grid5000 and DAS-2 type of workloads.
- Validation of the results obtained through expected value analysis.

The remaining part of the paper is organized as follows: Section 2 discusses the related works; Section 3 presents the system model considered for operation along with the definitions for the performance metrics; Section 4 presents the proposed architecture for load balancing containing the details about HMM view of grid resource model, Partially Observable Markov Decision Process (POMDP) view of client task model and T2SS based Dyna-Q-Learning Task Scheduling Framework; Section 5 presents the results and discussion; and finally Section 6 draws the conclusion.

2. Related work

Bhatia [11] provides a brief review of the task scheduling algorithms in HPC environment. The grid computing involves various research problems pertaining to load balancing, workflow scheduling, security, information management, and so on. The grids are broadly classified into two types that are computational grid and data grid. Computational grids are used to meet the scheduling demands of the complex scientific problems and data grids are used to meet the scheduling demands of substantial number of storage devices. The task scheduling is one of the potential problems in grid systems and the available scheduling algorithms in literature fall into three categories that are centralized, decentralized and hierarchical which operate either in online mode or offline mode. Several heuristic algorithms like genetic algorithm, simulated annealing, tabu search, swarm intelligence, reinforcement learning, opportunistic mechanism, suffrage, Max-Min, Min-Min, and many more are discussed in literature. All these algorithms find difficulty in exploring large state space due to poor uncertainty handling capacity which often leads to performance degradation.

Casagrande et al. [12] discuss deep reinforcement learning based job scheduler. Adaptive job scheduling policies that are energy efficient are formulated by efficiently exploiting the workload patterns in the grid systems. This incorporates deep learning approach inside the computational agent to make runtime job scheduling decisions through trial-and-error mechanism. The computational agent begins with no prior knowledge and starts to gain the reward which represents the quality of the action taken at each observed state. Complicated job control problems that arise within the large state space environment are managed efficiently. But it fails miserably to address specific characteristics of the workload on varying degree of sequential jobs.

Eng et al. [13] deal with task scheduling using hybrid approach that combines both variable neighborhood descent algorithm and great deluge algorithm. The vector-based representation of the tasks is performed. The heuristic tries to improve the solution by replacing the current solution with neighboring solution having higher fitness value. Multiple neighborhood search strategy is employed in a systematic manner to properly schedule independent tasks. The Deluge algorithm uses boundary value concept in a linear way throughout the search process to arrive at the high-quality job scheduling solutions. Combination of both heuristic algorithms produces global optimal policies, but the computational time taken to produce neighboring solutions is extremely high.

Umar and Pujiyanta [14] present ejecting based dynamic task scheduling in combination with the conventional First Come First Serve (FCFS) technique. The task arrival rate is assumed to follows Poisson distribution. Scheduling begins by placing the incoming tasks in a logical matrix using ejected based FCFS technique. Further inverse permutation matrix, transpose of the permutation matrix, and line vector of the tasks gets computed. The computation process is repeated in iterations until all the tasks are distributed efficiently among the grid resources. The execution time of the scheduler is less due to fast computation of the transpose of task matrix. However, the policy formulated is suboptimal as it fails to manage properly the varying workload pattern of the task.

Tang et al. [15] describe memory based homogeneous task scheduling technique. The distributed Particle Swarm Optimization (PSO) with memory function mimics the behavior of intelligent swarm groups to formulate scalable tasks scheduling policies. Initially, the tasks are submitted with the resource requirement into the waiting queue to get executed. The distributed particles in the grid space exchange their load information with their peers to determine the minimum loaded particle. Further tasks scheduling happens through exchange of load information between the nearby particles at a faster speed. However, the convergence rate is low as it gets trapped in local optimum solution under high uncertainty.

The limitations identified in the existing works are as follows [16, 17].

- Tailoring of system specific task scheduling policies often ends up in formulating sub-optimal policies.
- Existing policies are rule-based which fails to understand the exact workload pattern.
- Unable to determine candidate solutions which results in performance degradation.
- Improper modelling of hidden states and partial states results in poor task scheduling solutions.

3. System model

Consider a grid computing environment comprising of a set of incoming client tasks $\mathrm{CT} = \langle \mathrm{CT}_1, \mathrm{CT}_2, \mathrm{CT}_3, \dots, \mathrm{CT}_n \rangle$, every client task is identified by parameters that include parameter id (pid), resource requirement (rr), workload and speed of operation (sop) $\mathrm{CT}_i = \langle \mathrm{CT}_{\mathrm{id}}, \mathrm{CT}^{\mathrm{at}} = (\mathrm{pid}, \mathrm{rr}, \mathrm{sop}) \rangle$. T2SS of $\mathrm{CT}_i{}'s$ sare created such that for any pair of client task $(\mathrm{CT}_i,\ \mathrm{CT}^{\mathrm{at}})$ over universal set U there exists a mapping such that $\mathrm{CT}_i\colon \mathrm{C\,T}^{\mathrm{at}} \to P(U)$ which produces T2SS of tasks $\mathrm{T2SS\_CT} = \langle \mathrm{T2SS\_CT}_1, \dots, \mathrm{T2SS\_CT}_n \rangle$. T2SS of grid resources $\mathrm{GR}_i{}'s$ are created such that for any pair of grid resource $(\mathrm{GR}_i,\ \mathrm{GR}^{\mathrm{at}})$ over universal set U there exists a mapping such that $\mathrm{GR}_i\colon \mathrm{GR}^{\mathrm{at}} \to P(U)$which produces T2SS of grid resources $\mathrm{TT2SS\_GR} = \langle \mathrm{T2SS\_GR}_1, \mathrm{T2SS\_GR}_2, \dots, \mathrm{T2SS\_GR}_n \rangle$. The Dyna-Q-Learning scheduler performs action to map the $T2SS$ of client tasks among appropriate grid resources to receive maximum number of the rewards, i.e., $\mathrm{CT}_i * A_n \to R$. After sufficient iteration of training, the Dyna-Q-Learning scheduler develops an optimal task mapping policy $\pi_i = \mathrm{argmax}_a Q(\mathrm{CT}_i, a)$, $a_i \in A$. The mathematical definition for the Performance Metrics (PM) considered are four as follows.

PM1. Learning rate(LR(Dyna_QL))

Learning rate is the speed at which the Dyna-Q-Learning scheduler finds an optimal match between the T2SS of client tasks and the T2SS of grid resources. It is computed by considering the time taken to form first policy (t_π^1) and the time taken to form total number of policies (t_π^N).

(1) $$\mathrm{LR}(\mathrm{Dyna\_QL}) = (t_\pi^1 * t_\pi^N).$$

PM2. Accuracy (AC(Dyna_QL))

Accuracy of the Dyna-Q-Learning scheduler is the degree to which the outcome of the client task to grid resource mapping adheres to the standard of efficient completion of client tasks and intact use of the grid resources. It is the ratio of ideal mapping (OM) and the sum of Ideal and Non-Ideal Mapping (IM+NIM). In which *i* represents the number of client tasks and *j* represents number of grid resources.

(2) $$\mathrm{AC}\left(\mathrm{Dyna}_{\mathrm{QL}}\right) = \sum_{i=1}^{i=n} \sum_{j=1}^{j=k} \left[\frac{\mathrm{IM}_{ij}}{(\mathrm{IM}_{ij}+\mathrm{NIM}_{ij})}\right].$$

PM3. Execution time (ET(Dyna_QL))

Execution time of the Dyna-Q-Learning scheduler is the total time spent by the Dyna-Q-Learning scheduler to schedule the client tasks properly among the grid resources in grid resource pool. It is the sum of the times taken to create T2SS of client tasks $(\mathrm{T}_{\mathrm{T2SS\_CT}_i})$, grid resources $(T_{\mathrm{T2SS\_GR}_i})$, and tasks scheduling distribution policies (T_{π_i}).

(3) $$\mathrm{ET}(\mathrm{Dyna\_QL}) = \sum_{i=1}^{i=n}[T_{\mathrm{T2SS\_CT}_i}]+\sum_{i=1}^{i=k}[T_{\mathrm{T2SS\_GR}_i}]+\sum_{i=1}^{i=l}[T_{\pi_i}].$$

PM4. Resource utilization rate (RU(Dyna_QL))

The resource utilization rate of the Dyna-Q-Learning scheduler is the utilities of the grid resources that are consumed by the incoming client tasks. It is measured by computing the number of grid resources that were left idle $(N_{\mathrm{GR}^{i\mathrm{dle}}})$ and over-utilized $(N_{\mathrm{GR}_i^{i\mathrm{dle}}})$ among *N* available grid resources.

(4) $$\mathrm{AC}(\mathrm{Dyna\_QL})) = \left[N_{\mathrm{GR}_i^{\mathrm{over}}} - N_{\mathrm{GR}_i^{\mathrm{idle}}}\right]/N.$$

4. Proposed work

The proposed work discusses the HMM based grid resource model, POMDP based client task model and T2SS-based Dyna-Q-Learning Task Scheduling Framework. The states of the grid resources are unobservable hence flexible generalization of grid resources is required which is achieved via HMM representation. The states of the client tasks are partially observable and needs clear representation which is achieved via POMDP representation. The T2SS based Dyna-QL-Scheduler framework forms high quality task scheduling policies using Dyna reinforcement learning technique that can gain knowledge from simulated grid scenario by keeping track of each of state and action pair visited by the agent.

4.1. HMM view of the grid resource model

The resources in grid computing environment get affected due to uncertainty which is modelled using HMM [18]. Several causes for uncertainty are elasticity of resources, high-energy consumption, inconsistency of states, dynamic sharing of resources, poor coordination among resources, no clear standards, multi administration of organizations, constrained number of resources, non-determinism of resources, independent failures and so on. Due to the uncertainty in grid resources, the dynamics of the resource is completely hidden. Hence, the grid resources are modelled using HMM.

The HMM of grid resource is defined by five attributes.

(5) $\mathrm{HMM}(\mathrm{GR}_i) = \langle \mathrm{GR}_i, S(\mathrm{GR}_i), V(\mathrm{GR}_i), B(\mathrm{GR}_i), A(\mathrm{GR}_i), I(\mathrm{GR}_i)\rangle,\ \mathrm{GR}_i \in \mathrm{GR}$,

where:

$S(\mathrm{GR}_i)$ = Set of all hidden states of the grid resource, i.e., $|S(\mathrm{GR}_i)| = M$, where M is finite;

$V(\mathrm{GR}_i)$ = Set of observable symbols of the hidden state, i.e., $|\,V(\mathrm{GR}_i)| = N$, where N is finite

$B(\mathrm{GR}_i)$=Initial state probability of the grid resource;

$A(\mathrm{GR}_i) =$ State transition probabilities $A(\mathrm{GR}_i), A(\mathrm{GR}_i) \in\ S(\mathrm{GR}_i)$;

$I(\mathrm{GR}_i)$ =Probability emission matrix of the grid resource.

4.2. POMDP view of the client task model

The incoming client tasks also get affected by uncertainty which is modelled using POMDP [19]. Several factors that cause uncertainty are demand fluctuation, gaps in the security, inconsistent dataflow, variety of data, vastness of data, operational problems, exploration between the tasks, conflicting goals and so on. Due to the uncertainty in tasks, the exact dynamics of the tasks is not visible. Hence, the incoming tasks are modelled using POMDP model.

The POMDP of grid task is defined by five attributes.

(6)$\mathrm{POMDP}(\mathrm{CT}_i) = \langle \mathrm{CT}_i, S(\mathrm{CT}_i), A(\mathrm{CT}_i), P(\mathrm{CT}_i), R(\mathrm{CT}_i), \Omega(\mathrm{CT}_i), O(\mathrm{CT}_i)\rangle, \mathrm{CT}_i \in \mathrm{CT}$,

where:

$S(\mathrm{CT}_i)$ = Set of all possible partial states of the client task, i.e.,$|S(\mathrm{CT}_i)| = M$, where M is finite;

$A(\mathrm{CT}_i)$ = Set of all state actions;

$P(\mathrm{CT}_i)$= Transition function where $S(\mathrm{CT}_i) \times A(\mathrm{CT}_i) \times S(\mathrm{CT}_i) \to [0, 1]$;

$R(\mathrm{CT}_i)$=Reward function, i.e., $R(\mathrm{CT}_i) \leftarrow S(\mathrm{CT}_i) \times A(\mathrm{CT}_i)$;

$\Omega(\mathrm{CT}_i)$= Set of all partial observations;

$O(\mathrm{CT}_i) = \mathrm{Observation\ function}$, i.e., $O(\mathrm{CT}_i) = S(\mathrm{CT}_i) \times A(\mathrm{CT}_i) \times \Omega(\mathrm{CT}_i)$.

4.3. Proposed T2SS based Dyna-Q-Learning task scheduling framework

Fig. 1 gives pictorial representation of the proposed T2SS-based Dyna-Q-Learning task scheduling framework for grid computing environment. The framework is composed of three functional components that are T2SS-based Task Uncertainty reducer (T2SS_CTUR), T2SS-based Grid Resource Uncertainty reducer (T2SS_GRUR) and Dyna-Q-Learning-Scheduler (Dyna-QL-Scheduler). The Grid Information Service (GIS) stores the information about the grid resources in terms of resource id number, availability of the resource, capacity of the resource, type of the resource and current load over the grid resource. The T2SS_CTUR and T2SS_GRUR modules input incoming client tasks, grid resource parameters, and manage the uncertainty using T2SS. The Dyna-QL-Scheduler uses the Dyna-Q Algorithm that allows the scheduler to start learning and keep improving incrementally to form optimal task scheduling policies.

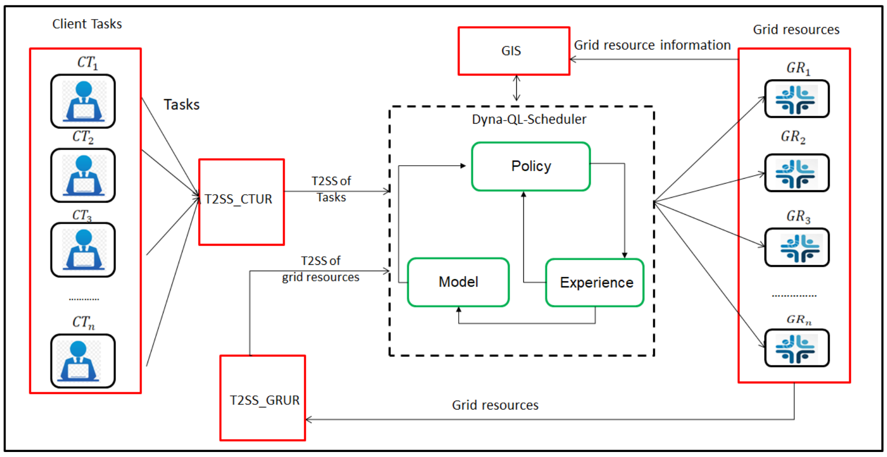


Fig. 1. Proposed T2SS based Dyna-Q-Learning task scheduling framework

4.3.1. T2SS based Client Task Uncertainty Reducer (T2SS_CTUR)

The T2SS_CTUR, inputs the client tasks with uncertainty $\mathrm{CT} = \langle \mathrm{CT}_1, \mathrm{CT}_2, \mathrm{CT}_3, \dots, \mathrm{CT}_n \rangle$ to generate T2SS of tasks, which are free from uncertainty $\mathrm{T2SS\_CT} = \langle \mathrm{T2SS\_CT}_1, \mathrm{T2SS\_CT}_2, \dots, \mathrm{T2SS\_CT}_n \rangle$, by considering wide range of client task attributes. The T2SS of client tasks over a universal set *U* and set of client task attributes $\mathrm{CT}^{\mathrm{at}}$ is represented as $F\left(\mathrm{CT}_i^{\mathrm{at}}\right)$, where $\mathrm{CT}_i^{\mathrm{at}}$ is the subset of $\mathrm{CT}^{\mathrm{at}}$and *F* is a function from $\mathrm{CT}_i^{\mathrm{at}}$to the power set of *U* *P*(*U*). The detail working of the T2SS_CTUR is given in Algorithm 1.

```
Algorithm 1. Working of T2SS_CTUR
Step 1. Begin
Step 2. Input: $\mathrm{CT} = \langle \mathrm{CT}_1, \mathrm{CT}_2, \mathrm{CT}_3, \dots, \mathrm{CT}_n \rangle$
Step 3. Output: $\mathrm{T2SS\_CT} = \langle \mathrm{T2SS\_CT}_1, \dots, \mathrm{T2SS\_CT}_n \rangle$
Step 4. Training: T2SS_CTUR
Step 5. for every training client task set CT do
Step 6.     for every ordered training client task pair
            $(\mathrm{CT}_i, \mathrm{CT}_i^{\mathrm{at}})$ in CT do
Step 7.         $\mathrm{CT}_i^{\mathrm{at}} \in \mathrm{CT}^{\mathrm{at}}$ , $F\left(\mathrm{CT}_i^{\mathrm{at}}\right) = \varnothing$
Step 8.         Compute training T2SS of tasks, i.e.,
            $\mathrm{T2SS\_CT}_i = \{F\left(\mathrm{CT}_i^{\mathrm{at}}\right)\} \in P(U)$
Step 9.     end for
Step 10. end for
Step 11. Testing: T2SS_CTUR
Step 12. for every client test task set CT do
Step 13.     for every ordered test client task pair $(\mathrm{CT}_i, \mathrm{CT}_i^{\mathrm{at}})$
            in CT do
Step 14.     Enumerate $\mathrm{T2SS\_CT} ::= \mathrm{T2SS\_CT}_i \cup (\mathrm{CT}_i, \mathrm{CT}_i^{\mathrm{at}})$
Step 15.     end for
Step 16.   end for
Step 17. Output T2SS of client tasks
             $\mathrm{T2SS\_CT} = \langle \mathrm{T2SS\_CT}_1, \mathrm{T2SS\_CT}_2, \dots, \mathrm{T2SS\_CT}_n \rangle$
Step 18. End
```

4.3.2. T2SS based Grid Resource Uncertainty Reducer (T2SS_GRUR)

The T2SS_GRUR inputs grid resource set $\mathrm{GR} = \langle \mathrm{GR}_1, \mathrm{GR}_2, \mathrm{GR}_3 \dots \mathrm{GR}_n \rangle$ to create T2SS of grid resources over a universal set U and a set of grid resource attributes $\mathrm{GR}_i^{\mathrm{at}}$ is represented as $F(\mathrm{GR}_i^{\mathrm{at}})$ where $\mathrm{GR}_i^{\mathrm{at}}$ is the subset of $\mathrm{GR}^{\mathrm{at}}$, and F is a function from $\mathrm{GR}_i^{\mathrm{at}}$ to the power set of U $P(U)$ to generate T2SS of grid resources $\mathrm{TT2SS\_GR} = \langle \mathrm{T2SS\_GR}_1, \mathrm{T2SS\_GR}_2, \dots, \mathrm{T2SS\_GR}_n \rangle$. The detail working of T2SS_GRUR is given in Algorithm 2.

```
Algorithm 2. Working of T2SS_GRUR
Step 1. Begin
Step 2. Input: $\mathrm{GR} = \langle \mathrm{GR}_1, \mathrm{GR}_2, \mathrm{GR}_3 \dots \mathrm{GR}_n \rangle$
Step 3. Output: $\mathrm{T2SS\_GR} = \langle \mathrm{T2SS\_GR}_1, \mathrm{T2SS\_GR}_2, \dots, \mathrm{T2SS\_GR}_n \rangle$
Step 4. Training: T2SS_GRUR
Step 5. for every training grid resource set GR do
Step 6.   for every ordered training grid resource pair, i.e., $(\mathrm{GR}_i, \mathrm{GR}_i^{\mathrm{at}})$ in GR do
Step 7.   $\mathrm{GR}_i^{\mathrm{at}} \in \mathrm{VM}_i^{\mathrm{at}}, F(\mathrm{GR}_i^{\mathrm{at}}) = \emptyset$
Step 8.       Compute training soft-set of grid resource pair
          $\mathrm{T2SS\_GR}_i = \{F(\mathrm{GR}_i^{\mathrm{at}})\} \in P(U)$
Step 9.      end for
Step 10.  end for
Step 11. Testing: T2SS_GRUR
Step 12. for every grid resource set GR do
Step 13. Use the test T2SS of grid resource, i.e., $\mathrm{T2SS\_GR}_i = \{F(\mathrm{GR}_i^{\mathrm{at}}) \in P(U)\}$
Step 14.   Enumerate $\mathrm{T2SS\_GR} ::= \mathrm{T2SS\_GR}_i \cup (\mathrm{GR}_i, \mathrm{GR}_i^{\mathrm{at}})$
Step 15.   end for
Step 16.  Output T2SS of grid resource, i.e.,
          $\mathrm{T2SS\_GR} ::= \langle \mathrm{T2SS\_GR}_1, \mathrm{T2SS\_GR}_2, \dots, \mathrm{T2SS\_GR}_n \rangle$
Step 17. End
```

4.3.3. Dyna-QL-Scheduler

The Dyna-QL-Scheduler inputs T2SS of client tasks $\mathrm{T2SS\_CT} = \langle \mathrm{T2SS\_CT}_1, \dots, \mathrm{T2SS\_CT}_n \rangle$ and grid resources $\mathrm{T2SS\_GR} = \langle \mathrm{T2SS\_GR}_1, \dots, \mathrm{T2SS\_GR}_n \rangle$ to map the client tasks to appropriate grid resources by forming optimal task scheduling policy $\mathrm{CT}_i \times A_n \to R$ using Dyna-Q-Learning Algorithm. The Dyna-Q-Learning Algorithm combines both direct reinforcement learning and model-based learning to perform one step tabular Q-Learning for planning phase and one step tabular Q-Learning learning phase. The detail working of the Dyna-QL-Scheduler is given in Algorithm 3.

```
Algorithm 3. Working of Dyna-QL-Scheduler
Step 1. Begin
Step 2. Input: $\mathrm{T2SS\_CT} = \langle \mathrm{T2SS\_CT}_1, \dots, \mathrm{T2SS\_CT}_n \rangle$ and
                $\mathrm{T2SS\_GR} = \langle \mathrm{T2SS\_GR}_1, \dots, \mathrm{T2SS\_GR}_n \rangle$
Step 3. Output:
                $\Pi(\mathrm{T2SS\_CT} \rightarrow \mathrm{T2SS\_GR}) = \langle \Pi_1, \Pi_2, \Pi_3, \dots, \Pi_p \rangle$
Step 4. Training: Dyna-QL-Scheduler
Step 5. Initialize $Q(S, A) = \emptyset$ and $\mathrm{Model}(S, A) = \emptyset$ for all $s \in$ Sanda $\in A$
Step 6. for every training $\mathrm{T2SS\_CT}_i$ and $\mathrm{T2SS\_GR}_i$ do
Step 7. Begin Q-Learning phase
Step 8.    Assign the state $S \leftarrow$ current non-terminal state
Step 9.    Assign the action $A \leftarrow \epsilon$ greedy policy$(S, A)$
Step 10.    Take an action a, where $a \in A$, observe the resultant reward $R$ and
update the state $S \rightarrow S^1$
Step 11.    Compute the $Q$-value
        $Q(S, A) = Q(S, A) + \alpha[R + \gamma \max_a Q(S^1, A) - Q(S, A)]$
Step 12. End Q-Learning training phase
Step 13. Begin Model learning training phase
Step 14.    Assume the grid computing environment is deterministic after
applying T2SS theory over the tasks and grid resources
Step 15.    Update the training model $\mathrm{Model}(S, A) \leftarrow R$ and
            $\mathrm{Model}(S, A) \leftarrow S^1$
Step 16. End Model learning training phase
Step 17. Begin planning training phase
Step 18.    Assign the state $S \leftarrow$previously observed random state
Step 19.    Assign the action $A \leftarrow$Random action take in the state $S$
Step 20.    Compute reward $R$, Updated state $S^1$, i.e.,
          R, $S^1 \leftarrow \mathrm{Model}(S, A)$
Step 21.    Compute $Q(S, A) = Q(S, A) + \alpha[R + \gamma \max_a Q(S^1, A) - Q(S, A)]$
Step 22. End planning training phase
Step 23. Formulate policies $\Pi ::= \Pi \cup \Pi_i$
Step 24. Testing: Dyna-QL-Scheduler
Step 25. Initialize $Q(S, A) = \emptyset$ and $\mathrm{Model}(S, A) = \emptyset$ for all $s \in$ Sanda $\in A$
Step 26. for every testing $\mathrm{T2SS\_CT}_i$ and $\mathrm{T2SS\_GR}_i$ do
Step 27. Begin Q-Learning testing phase
Step 28.    Take an action a, where $a \in A$, observe the resultant reward $R$ and
          update the state $S \rightarrow S^1$
Step 29.    Compute the $Q$ value
        $Q(S, A) = Q(S, A) + \alpha[R + \gamma \max_a Q(S^1, A) - Q(S, A)]$
Step 30. End Q-Learning testing phase
Step 31. Begin Model learning testing phase
Step 32.    Update the testing model $\mathrm{Model}(S, A) \leftarrow R$ and $\mathrm{Model}(S, A) \leftarrow S^1$
Step 33. End Model learning testing phase
Step 34. Begin planning testing phase
Step 35.    Compute reward $R$, Updated state $S^1$, i.e., R, $S^1 \leftarrow \mathrm{Model}(S, A)$
```

Step 36. Compute $Q(S, A) = Q(S, A) + \alpha[R + \gamma \max_a Q(S^1, A) - Q(S, A)]$

Step 37. End planning testing phase

Step 38. Output the optimal T2SS of client tasks to grid resource mapping policies, i.e., $\Pi = \langle \Pi_1, \Pi_2, \dots, \Pi_\mathrm{p} \rangle$

Step 39. End

5. Results

The simulation of the proposed Dyna-QL-Scheduler is evaluated using GridSim Alea simulator on Lenovo Aspire3 Quadcore machine with 2058 MB of RAM. The performance of the proposed Dyna-QL-Scheduler is compared with existing schedulers by considering with two types of workloads that are BASIC and EXTENDED (EXT). Under EXTENDED workload three distinct types are considered that are EXT_FAIL, EXT_REQ and EXT_ALL [22, 23]. For EXT_FAIL workload, higher priority is given for failure of the grid devices and specific requirements of the client tasks are ignored which are managed by EXTENDED workload. For EXT_REQ workload, the failure of the grid devices is ignored, and higher priority is given for requirements of the client tasks. For EXT_ALL workload both failure of grid devices and particular requirements of the client tasks are given higher priority. Different experiments are conducted by considering MetaCentrum, Grid5000 and DAS-2 problem scenarios. The DAS-2 workload has two different variants of grid devices which are referred as DAS-2-L and DAS-2-M. Table 1 gives the summary of the workloads, problems and experiments conducted details [24, 25].

Table 1. Summary of workloads, problems and experiments using GridSim

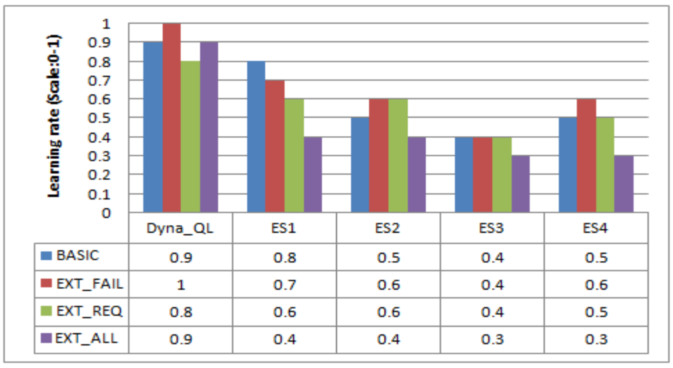
Workloads	Problem scenarios			
MetaCentrum	BASIC	EXT_FAIL	EXT_REQ	EXT_ALL
Grid5000	BASIC	EXT_FAIL	EXT_REQ	EXT_ALL
DAS2	BASIC	EXT_FAIL_L	EXT_FAIL_M	EXT_REQ
	EXT_ALL_L	EXT_ALL_M		

5.1. Experiment 1. MetaCentrum workload

The MetaCentrum workload with BASIC, EXT_FAIL, EXT_REQ and EXT_ALL scenarios are taken into consideration. Comparison of the Dyna-QL-Scheduler against four existing schedulers ES1, ES2, ES3, and ES4 is conducted towards performance metrics like learning rate, accuracy, execution time and resource utilization rate.

5.1.1. Learning rate

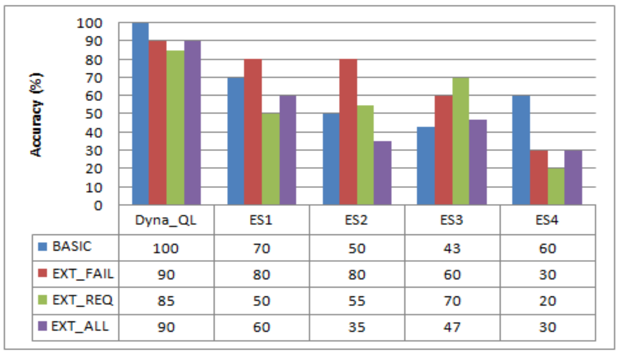
Fig. 2 gives the comparison over the learning rate. It is observed from the graph that the learning rate incurred by the Dyna-QL-Scheduler is high for all the four types of problem scenarios as the model learns from experience instead of performing random actions. The learning rate incurred by the ES1 is above average for BASIC, EXT_FAIL, EXT_REQ but the learning rate is below average for EXT_ALL as the model requires more data for processing and suffers from overfitting problem due to intense training. The learning rate incurred by the ES2 is average for EXT_FAIL, EXT_REQ but the learning rate is below average for BASIC and EXT_ALL as the model is capable enough of handling large state space but suffers from optimization problem. The learning rate of ES3 is poor for all the four types of problem scenarios as the lengthier tasks suffer from starvation. The learning rate incurred by the ES4 is above average for Dyna-QL, EXT_FAIL, and EXT_REQ but the learning rate is poor for EXT_ALL as the model easily gets trapped into local optimum solution.



	Dyna_QL	ES1	ES2	ES3	ES4
🟦 BASIC	0.9	0.8	0.5	0.4	0.5
🟥 EXT_FAIL	1	0.7	0.6	0.4	0.6
🟩 EXT_REQ	0.8	0.6	0.6	0.4	0.5
🟪 EXT_ALL	0.9	0.4	0.4	0.3	0.3

Fig. 2. Learning rate incurred for MetaCentrum workload

5.1.2. Accuracy



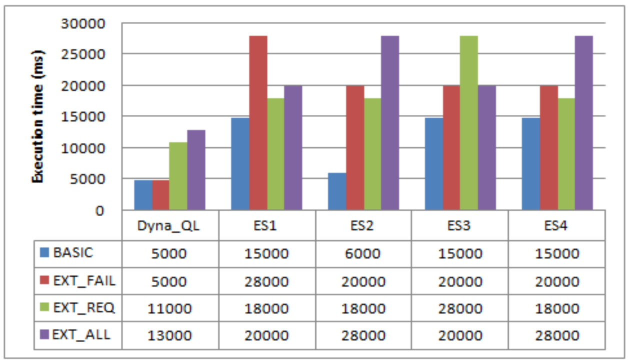
	Dyna_QL	ES1	ES2	ES3	ES4
🟦 BASIC	100	70	50	43	60
🟥 EXT_FAIL	90	80	80	60	30
🟩 EXT_REQ	85	50	55	70	20
🟪 EXT_ALL	90	60	35	47	30

Fig. 3. Accuracy incurred for MetaCentrum workload

Fig. 3 gives the comparison over the accuracy achieved. It is observed from the graph that the accuracy achieved by the Dyna-QL-Scheduler is high for all the four types of problem scenarios as the model stays stable when exposed to uncertainty. The accuracy achieved by the ES1 is above average for BASIC, EXT_FAIL, EXT_ALL but the accuracy is below average for EXT_REQ as the chances of the model forgetting previous learnt knowledge is more when exposed to large state space problems. The accuracy achieved by the ES2 is average for BASIC, EXT_REQ and EXT_all but above average for EXT_FAIL as the heuristic function of the model suddenly drops when exposed to critical situation. The accuracy achieved by the ES3 is average for EXT_FAIL and EXT_REQ but it is poor for BASIC and EXT_ALL as the model neglect the tasks with late due dates. The accuracy achieved by the ES4 is average for BASIC but very less for EXT_REQ, EXT_FAIL and EXT_ALL as the model fails to efficiently explore the workspace.

5.1.3. Execution time

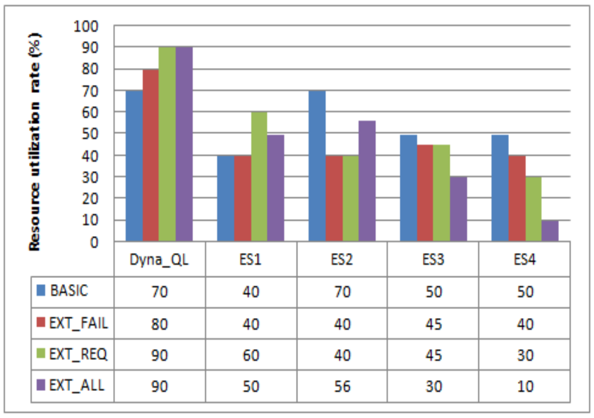
Fig. 4 gives the comparison over the execution time. It is observed from the graph that the execution time of the Dyna-QL-Scheduler is extremely low for BASIC and EXT_FAIL and below average for EXT_REQ and EXT_ALL as the model construct the Markov Decision Process (MDP) from experience at each computing stage to manage task dynamics. The execution time of the ES1 is average for BASIC, EXT_REQ and EXT_ALL but extremely high for EXT_FAIL as the computing states get overloaded due to too much of reinforcements. The execution time of the ES2 is low for BASIC but extremely high for EXT_REQ, EXT_FAIL and EXT_ALL as the model moves to worst state during termination. The execution time of the ES3 and ES4 are very high for EXT_REQ, EXT_FAIL and EXT_ALL but less for BASIC as the model take a greater number of iterations to converge towards promising solutions.



	Dyna_QL	ES1	ES2	ES3	ES4
🟦 BASIC	5000	15000	6000	15000	15000
🟥 EXT_FAIL	5000	28000	20000	20000	20000
🟩 EXT_REQ	11000	18000	18000	28000	18000
🟪 EXT_ALL	13000	20000	28000	20000	28000

Fig. 4. Execution time incurred for MetaCentrum workload

5.1.4. Resource utilization rate



	Dyna_QL	ES1	ES2	ES3	ES4
🟦 BASIC	70	40	70	50	50
🟥 EXT_FAIL	80	40	40	45	40
🟩 EXT_REQ	90	60	40	45	30
🟪 EXT_ALL	90	50	56	30	10

Fig. 5. Resource utilization rate incurred for MetaCentrum workload

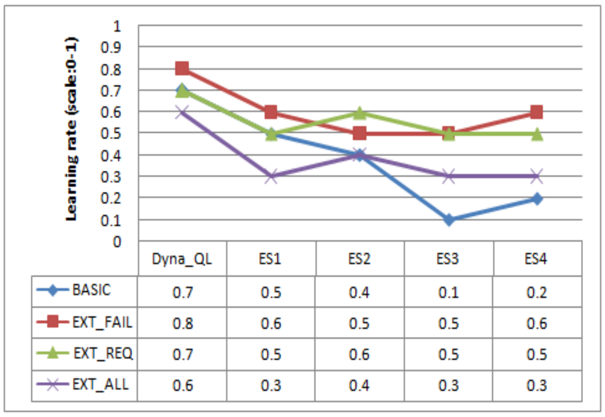
Fig. 5 gives the comparison over the resource utilization rate. It is observed from the graph that the resource utilization rate of the Dyna-QL-Scheduler is extremely high for all the four types of problem scenarios as the model is capable enough of accumulating with minimum epochs of planning. The resource utilization rate of the

ES1 is low for BASIC and EXT_REQ and is found to be average for EXT_FAIL and EXT_ALL as the model does not have standard for training and tuning. The resource utilization rate of the ES2 of high for BASIC and EXT_ALL and is found to be low for EXT_FAIL and EXT_REQ as the model repeatedly changes the neighborhood position and fails to obtain exact valley results during perturbation phase. The resource utilization rate of the ES3 is below average for all the four types of scenarios as the model lacks scalability and suffers from non-trivial verifiability problem. The resource utilization rate of the ES4 is below average for BASIC, EXT_REQ, EXT_FAIL and extremely low for EXT_ALL due to poor clamping of PSO particles during planning and modelling phases of the algorithm.

5.2. Experiment 2. Grid5000 workload

The Grid5000 workload with BASIC, EXTENDED_FAIL, EXTENDED_REQ and EXTENDED_ALL problem scenarios are considered. Comparison of the Dyna-QL-Scheduler against four existing schedulers ES1, ES2, ES3, and ES4 is conducted towards performance metrics like learning rate, accuracy, execution time and resource utilization rate.

5.2.1. Learning rate



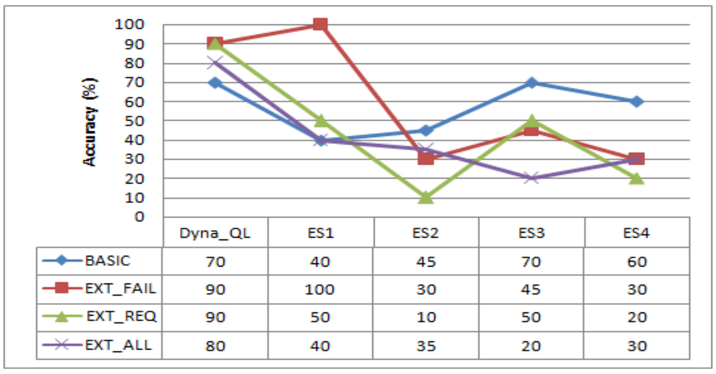
	Dyna_QL	ES1	ES2	ES3	ES4
BASIC	0.7	0.5	0.4	0.1	0.2
EXT_FAIL	0.8	0.6	0.5	0.5	0.6
EXT_REQ	0.7	0.5	0.6	0.5	0.5
EXT_ALL	0.6	0.3	0.4	0.3	0.3

Fig. 6. Learning rate incurred for Grid5000 workload

Fig. 6 gives the comparison over the learning rate. It is observed from the graph that the learning rate of the Dyna-QL-Scheduler remained higher for all the four types of problem scenarios as the Q-Learning model does not need to compute separately the reward function or transition function. The learning rate of the ES1 is average for BASIC, EXT_FAIL and EXT_REQ but extremely low for EXT_ALL as the model is flexible enough to be optimally get tuned for desired solutions. The learning rate of the ES3 remained average for all the four types of problem scenarios as the model is fair for longer tasks and unfair for shorter tasks. The learning rate of the ES4 is average for EXT_FAIL and EXT_REQ and low for BAISC and EXT_ALL. The learning rate of the ES4 remained average for all the four types of problem scenarios as it gets easily stuck in local optima.

5.2.2. Accuracy

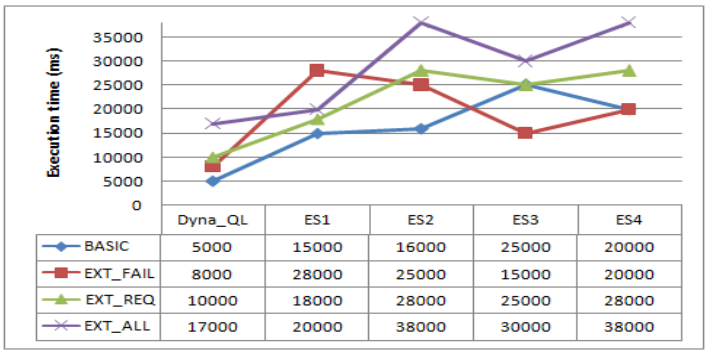
Fig. 7 gives the comparison over the accuracy achieved. It is observed from the graph that the accuracy of the Dyna-QL-Scheduler remained above average for all the four types of problem scenarios. The accuracy of the ES1 remained extremely high for EXT_FAIL but exceptionally low for BASIC, EXT_REQ and EXT_ALL as the model continuously gain knowledge from real-time experience and keep updating the Q-table. The accuracy of the ES2 is incredibly low for all four types of problem scenarios as the model employs harmony search to arrive at promising solutions which suffers from premature convergence to poor solutions. The accuracy of the ES3 remained above average for BASIC, EXT_FAIL and EXT_REQ but it is low for EXT_ALL as the model completely ignores the complexity of the data transmission. The accuracy of ES4 is above average for BASIC and exceptionally low for EXT_FAIL, EXT_REQ and EXT_ALL as the model fails to accurately compute the step size to determine global best particle.



	Dyna_QL	ES1	ES2	ES3	ES4
◆ BASIC	70	40	45	70	60
■ EXT_FAIL	90	100	30	45	30
▲ EXT_REQ	90	50	10	50	20
✕ EXT_ALL	80	40	35	20	30

Fig. 7. Accuracy incurred for Grid5000 workload

5.2.3. Execution time



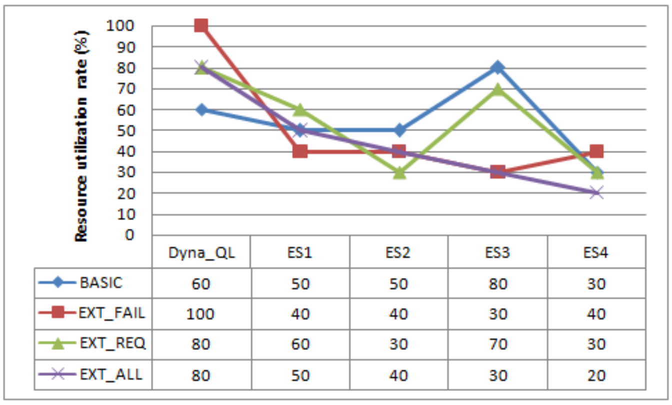
	Dyna_QL	ES1	ES2	ES3	ES4
◆ BASIC	5000	15000	16000	25000	20000
■ EXT_FAIL	8000	28000	25000	15000	20000
▲ EXT_REQ	10000	18000	28000	25000	28000
✕ EXT_ALL	17000	20000	38000	30000	38000

Fig. 8. Execution time incurred for Grid5000 workload

Fig. 8 gives the comparison for the execution time. The execution time of the Dyna-QL-Scheduler remained low for all the four types of problem scenarios as the policies formulated are of high quality due to proper optimization of reinforcement policies. The execution time of ES1 is moderate for all four types of problem scenarios due to the inherent limitation of policy gradient theorem. The execution time of ES2 is moderate for BASIC and EXT_FAIL but extremely high for EXT_REQ and EXT_ALL as the model cannot widen the search process to find out the promising solutions. The execution time of ES3 is high for all the four types of problem scenarios as the model demands for a greater number of computation cycles to reduce the tardiness of the tasks. The execution time of ES4 is very high for all the four types of problem scenarios as the model does not allow the logistic function to change the parameter values to overcome instability in learning.

5.2.4. Resource utilization rate

Fig. 9 gives the comparison of the resource utilization rate of the Dyna-QL-Scheduler with other four recent schedulers ES1, ES2, ES3 and ES4. The resource utilization rate of Dyna-QL-Scheduler is above average for all the four types of problem scenarios as the model require moderate number of experience tuples for convergence. The resource utilization rate of ES1 is average for all the four types of problem scenarios as the model requires moderate number of random search experiences to arrive at promising solutions. The resource utilization rate of ES3 is above average for BASIC and EXT_REQ but low for EXT_FAIL, and EXT_ALL as the smaller tasks gets ignored while processing larger tasks. The resource utilization rate of ES4 is low for all the four types of problem scenarios due to lack of precision factor.



	Dyna_QL	ES1	ES2	ES3	ES4
BASIC	60	50	50	80	30
EXT_FAIL	100	40	40	30	40
EXT_REQ	80	60	30	70	30
EXT_ALL	80	50	40	30	20

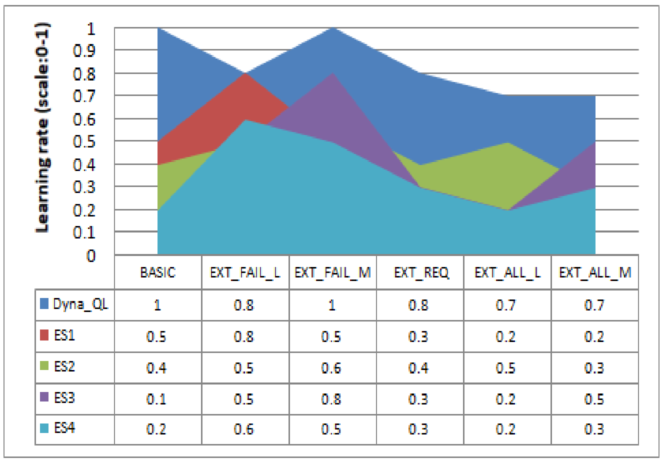
Fig. 9. Resource utilization rate for Grid5000 workload

5.3. Experiment 3. DAS-2 workload

The DAS-2 workload with BASIC, EXT_FAIL_L, EXT_FAIL_M, EXT_REQ, EXT_ALL_L and EXT_ALL_M problem scenarios are considered. Comparison of the Dyna-QL-Scheduler against four existing schedulers ES1, ES2, ES3, and ES4 is conducted towards performance metrics like learning rate, accuracy, execution time and resource utilization rate.

5.3.1. Learning rate

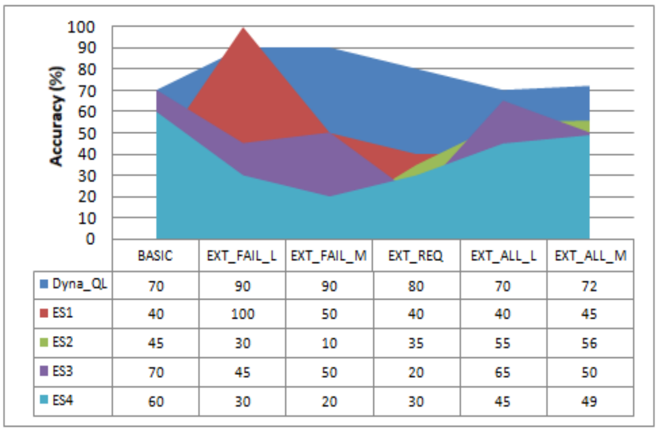
Fig. 10 gives the comparison for the learning rate attained. The learning rate of the Dyna-QL-Scheduler is extremely high for BASIC, EXT_FAIL_L, EXT_FAIL_M and EXT_REQ but average for EXT_ALL_L and EXT_ALL_M because the model converges to optimal policy with minimum exploitation of Q-Value. The learning rate of ES1isabove average for BASIC, EXT_FAIL_L and EXT_FAIL_M but low for other scenarios as the model becomes terribly unstable when exposed to larger state space problems. The learning rate of ES2 is below average for all six types of problem scenarios due to abrupt terminating of the algorithm when it reaches ridge positions in the larger state space. The learning rate of ES3 is high for EXT_FAIL_M but low for other problem scenarios as the model ignores disk service time. The learning rate of ES4 is high for EXT_FAIL_L and EXT_FAIL_M but low for other problem scenarios as convergence rate of the algorithm gradually decreases during particle evolution.



	BASIC	EXT_FAIL_L	EXT_FAIL_M	EXT_REQ	EXT_ALL_L	EXT_ALL_M
🟦 Dyna_QL	1	0.8	1	0.8	0.7	0.7
🟥 ES1	0.5	0.8	0.5	0.3	0.2	0.2
🟩 ES2	0.4	0.5	0.6	0.4	0.5	0.3
🟪 ES3	0.1	0.5	0.8	0.3	0.2	0.5
🟦 ES4	0.2	0.6	0.5	0.3	0.2	0.3

Fig. 10. Learning rate incurred for DAS-2 workload

5.3.2. Accuracy



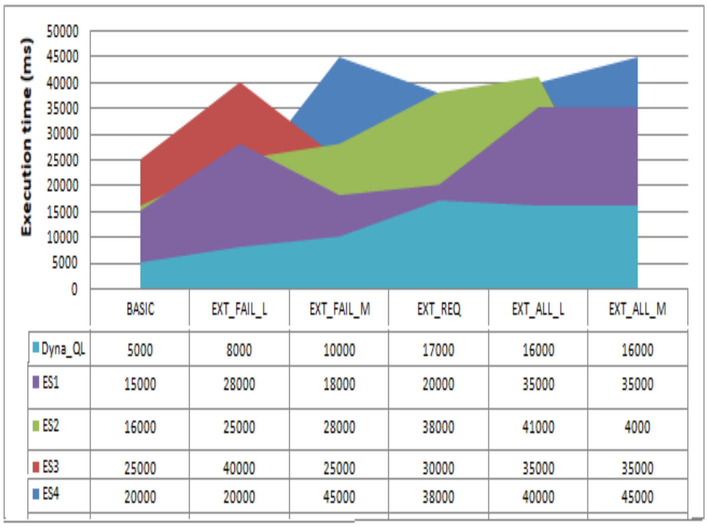
	BASIC	EXT_FAIL_L	EXT_FAIL_M	EXT_REQ	EXT_ALL_L	EXT_ALL_M
🟦 Dyna_QL	70	90	90	80	70	72
🟥 ES1	40	100	50	40	40	45
🟩 ES2	45	30	10	35	55	56
🟪 ES3	70	45	50	20	65	50
🟦 ES4	60	30	20	30	45	49

Fig. 11. Accuracy incurred forDAS-2 workload

Fig. 11 gives the comparison for the accuracy achieved. The accuracy achieved by the Dyna-QL-Scheduler is extremely high for all six types of scenarios due to deliberative planning of the Q-Learning agent in background at every time step of planning and modelling. The accuracy achieved by the ES1 is high for EXT_FAIL_L but low for other problem scenarios as the model suffers from catastrophic forgetting due to continuous learning process. The accuracy achieved by the ES2 is below average for all six types of scenarios as the steepest descent lacks monotone convergence. The accuracy achieved by the ES3 is higher for BASIC, EXT_FAIL_M and EXT_ALL_L type of scenarios but lower for EXT_FAIL_L, EXT_REQ and EXT_ALL_M as it ignores input/output efficiency and involves lot of random head movements towards suboptimal solutions. The accuracy achieved by the ES4 is low for all six types of scenarios.

5.3.3. Execution time

Fig. 12 gives the comparison for the execution time incurred. The execution time of the Dyna-QL-Scheduler remained low for all the six types of problem scenarios as the model gradually keeps learning by taking instant feedbacks and revises the task scheduling decisions. The execution time of ES1 is moderate for all six types of problem scenarios as the model is not scalable and often results in wrong solutions due to rotation invariants. The execution time of ES2 is moderate for all six types of problem scenarios as the model act decently when the search space is flat with same neighboring state values. The execution time of ES3 is moderate for all six types of problem scenarios as the model saturates the system when the load increases. The execution time of ES4 is extremely high for all six types of problem scenarios as the fitness functions are non-differentiable when exposed to large state space environment.

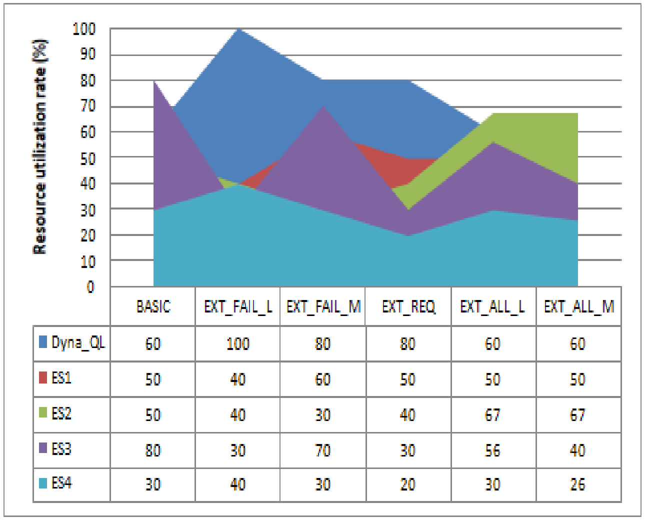


	BASIC	EXT_FAIL_L	EXT_FAIL_M	EXT_REQ	EXT_ALL_L	EXT_ALL_M
🟦 Dyna_QL	5000	8000	10000	17000	16000	16000
🟪 ES1	15000	28000	18000	20000	35000	35000
🟩 ES2	16000	25000	28000	38000	41000	4000
🟥 ES3	25000	40000	25000	30000	35000	35000
🟦 ES4	20000	20000	45000	38000	40000	45000

Fig. 12. Execution time incurred for DAS-2 workload

5.3.4. Resource utilization rate

Fig. 13 gives the comparison for the resource utilization rate. The resource utilization rate of Dyna-QL-Scheduler is high for all six types of problem scenarios as the model can efficiently handle stochastic transitions without requiring much adaptation. The resource utilization rate of ES1 and is average for all six types of scenarios as the model moderately scales well with the increase in data. The resource utilization rate of ES2 is above average for EXT_ALL_L and EXT_ALL_M type of scenarios but low for other problem scenarios as initial approximation of the solution is poor in terms of quality. The resource utilization rate of ES4 is below average for all six types of scenarios as the tendency of the model locating local minima is more instead of global minima.



	BASIC	EXT_FAIL_L	EXT_FAIL_M	EXT_REQ	EXT_ALL_L	EXT_ALL_M
🟦 Dyna_QL	60	100	80	80	60	60
🟥 ES1	50	40	60	50	50	50
🟩 ES2	50	40	30	40	67	67
🟪 ES3	80	30	70	30	56	40
🟦 ES4	30	40	30	20	30	26

Fig. 13. Resource utilization rate incurred for DAS-2 workload

The main limitation of the proposed Dyna-Q-Learning task scheduling framework in grid is occasional instability in the computing capability under dynamic variation in the workload. The grid devices with different variants pose several challenges at runtime in terms of sudden drop in processing capability, shortage of memory for memory intensive applications, and improper sharing of resource at components level. Here the challenges posed by grid devices variants during runtime are not handled in global optimal manner at runtime. Instead the Dyna-Q-Learning agent tries to converge to nearly optimal optimization policies.

6. Conclusion

The paper presents a novel Dyna-Q-Learning task scheduling framework. The uncertainties in the computing environment are precisely identified using appropriate Markov models. The proposed Dyna-Q-Learning task scheduler learns from experience and converges to optimal scheduling policies with minimum exploitation of Q-Value. From expected value analysis and simulation results, it is found that the proposed scheduler outperforms four recent schedulers with respect to performance parameters like learning rate, accuracy, execution time and resource utilization rate. We plan to extend the proposed scheduler with respect to system stability by establishing timing constraint for the scheduler at runtime.

Data availability. To carry out simulation of the proposed "Dyna-Q-Learning Framework for Task Scheduling in Grid Computing" three different workloads are considered that are MetaCentrum workload, Grid5000, and DAS2. The data is shown from two papers [22, 26].

References

1. Qasaimeh, M., R. S. Al-Qassas, S. Aljawarneh. Recent Development in Smart Grid Authentication Approaches: A Systematic Literature Review. – Cybernetics and Information Technologies, Vol. **19**, 2019, No 1, pp. 27-52.
2. Dabrowski, C. Reliability in Grid Computing Systems. – Concurrency and Computation: Practice and Experience, Vol. **21**, 2009, No 8, pp. 927-959.
3. Sadashiv, N., S. D. Kumar. Cluster, Grid and Cloud Computing: A Detailed Comparison. – In: Proc. of 6th International Conference on Computer Science & Education (ICCSE'11), IEEE, August 2011, pp. 477-482.
4. Casanova, H. Distributed Computing Research Issues in Grid Computing. – ACM SIGAct News, Vol. **33**, 2002, No 3, pp. 50-70.
5. Yu, J., R. Buyya, K. Ramamohanarao. Workflow Scheduling Algorithms for Grid Computing. – In: Proc. of Metaheuristics for Scheduling in Distributed Computing Environments, 2008, Berlin, Heidelberg, Springer, pp. 173-214.
6. Maji, P. K., R. Biswas, A. R. Roy. Soft Set Theory. – Computers & Mathematics with Applications, Vol. **45**, 2003, No 4-5, pp. 555-562.
7. Hayat, K., M. I. Ali, B. Y. Cao, X. P. Yang. A New Type-2 Soft Set: Type-2 Soft Graphs and Their Applications. – Advances in Fuzzy Systems, 2017.
8. Gu, S., T. Lillicrap, I. Sutskever, S. Levine. Continuous Deep q-Learning with Model-Based Acceleration. – In: Proc. of International Conference on Machine Learning, PMLR, June 2016, pp. 2829-2838.
9. Jeaunita, T. J., V. Sarasvathi. A Multi-Agent Reinforcement Learning-Based Optimized Routing for QoS in IoT. – Cybernetics and Information Technologies, Vol. **21**, 2021, No 4, pp. 45-61.
10. Eng, K., A. Muhammed, M. A. Mohamed, S. Hasan. A Hybrid Heuristic of Variable Neighbourhood Descent and Great Deluge Algorithm for Efficient Task Scheduling in Grid Computing. – European Journal of Operational Research, Vol. **284**, 2020, No 1, pp. 75-86.
11. Bhatia, M. K. Task Scheduling in Grid Computing: A Review. – Advances in Computational Sciences and Technology, Vol. **10**, 2017 No 6, pp. 1707-1714.
12. Casagrande, L. C., G. P. Koslovski, C. C. Miers, M. A. Pillon. DeepScheduling: Grid Computing Job Scheduler Based on Deep Reinforcement Learning. – In: Proc. of International Conference on Advanced Information Networking and Applications, April 2020, Springer Cham, pp. 1032-1044.
13. Eng, K., A. Muhammed, M. A. Mohamed, S. Hasan. A Hybrid Heuristic of Variable Neighbourhood Descent and Great Deluge Algorithm for Efficient Task Scheduling in Grid Computing. – European Journal of Operational Research, Vol. **284**, 2020, No 1, pp. 75-86.
14. Umar, R., A. Pujiyanta. Development of First Come First Serve-Ejecting Based Dynamic Scheduling (FCFS-EDS) Simulation Scheduling Method for MPI Job in a Grid System. – Journal of Engineering and Applied Sciences, Vol. **12**, 2017, No 8, pp. 1972-1978.
15. Tang, K., W. Jiang, R. Cui, Y. Wu. A Memory-Based Task Scheduling Algorithm for Grid Computing Based on Heterogeneous Platform and Homogeneous Tasks. – International Journal of Web and Grid Services, Vol. **16**, 2020, No 3, pp. 287-304.
16. Zeigler, B. P., A. Muzy, E. Kofman. Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations. Academic Press, 2018.
17. Zhang, J., G. Ding, Y. Zou, S. Qin, J. Fu. Review of Job Shop Scheduling Research and Its New Perspectives under Industry 4.0. – Journal of Intelligent Manufacturing, Vol. **30**, 2019, No 4, pp. 1809-1830.

18. N i e, R., S. H e, F. L i u, X. L u a n, H. S h e n. Hmm-Based Asynchronous Controller Design of Markovian Jumping Lur'e Systems within a Finite-Time Interval. – IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2020.
19. B h a t t a c h a r y a, S., S. B a d y a l, T. W h e e l e r, S. G i l, D. B e r t s e k a s. Reinforcement Learning for POMDP: Partitioned Rollout and Policy Iteration with Application to Autonomous Sequential Repair Problems. – IEEE Robotics and Automation Letters, Vol. **5**, 2020, No 3, pp. 3967-3974.
20. H e a t h, A., N. K u n s t, C. J a c k s o n, M. S t r o n g, F. A l a r i d-E s c u d e r o, J. D. G o l d h a b e r-F i e b e r t, H. J a l a l. Calculating the Expected Value of Sample Information in Practice: Considerations from 3 Case Studies. – Medical Decision Making, Vol. **40**, 2020, No 3, pp. 314-326.
21. H i r o n a k a, T., M. B. G i l e s, T. G o d a, H. T h o m. Multilevel Monte Carlo Estimation of the Expected Value of Sample Information. – SIAM/ASA Journal on Uncertainty Quantification, Vol. **8**, 2020, No 3, pp. 1236-1259.
22. K l u s a c e k, D., M. S o y s a l, F. S u t e r. Alea-Complex Job Scheduling Simulator. – In: 13th International Conference on Parallel Processing and Applied Mathematics, September 2019.
23. K a d a, B., H. K a l l a. An Efficient Fault-Tolerant Scheduling Approach with Energy Minimization for Hard Real-Time Embedded Systems. – In: Proc. of International Workshop on Distributed Computing for Emerging Smart Networks, October 2019, Springer Cham., pp. 102-117.
24. T o s h e v, A. Particle Swarm Optimization and Tabu Search Hybrid Algorithm for Flexible Job Shop Scheduling Problem-Analysis of Test Results. – Cybernetics and Information Technologies, Vol. **19**, 2019, No 4, pp. 26-44.
25. I v a n o v a-R o h l i n g, V. N., N. R o h l i n g. Evaluating Machine Learning Approaches for Discovering Optimal Sets of Projection Operators for Quantum State Tomography of Qubit Systems. – Cybernetics and Information Technologies, Vol. **20**, 2020, No 6, pp. 61-73.
26. E l e l i e m y, A., A. M o h a m m e d, F. M. C i o r b a. Exploring the Relation between Two Levels of Scheduling Using a Novel Simulation Approach. – In: Proc. of 16th International Symposium on Parallel and Distributed Computing (ISPDC'17), 2017, pp. 26-33.

Received: 25.03.2022; Second Version: 10.07.2022; Accepted: 28.07.2022