Scheduling Job Based on Priority Allocation With Machine Variability

R.Nithya
Dept of M.E
A.R.J College of Engineering and Technology

Abstract- *Research focus on today's large-scale data analytics, approximation jobs that allow partial execution of their many tasks to achieve valuable results have played a significant role. This fact can be utilized to maximize the system utility of a big data computing cluster by choosing proper tasks in scheduling for each approximation job. The job execution can be based on the dependencies of the each job. Each and every job contains individual sub tasks. Each job execution based on the dependencies of the other job. The independent of the job can be executed first and then other job can be executed. The dependencies can be categorized into two types one is direct dependencies and other one is indirect dependencies. The proposed method can be allocating the resource can be based on the dependencies and the particular job execution and its weight of the each job. The job can be either it is single job or batch based job.*

Keywords- Job Scheduling, performance evaluation, mean-field limit.

I. INTRODUCTION

Over the past decade, big data computing clusters with hundreds or thousands of servers have become increasingly common. Such data-parallel clusters usually deploy highly scalable computing frameworks like Map Reduce and Apache Tez to run data-intensive parallel jobs for improving utilization and cost efficiency. The performance of parallel jobs is often constrained by the cluster’s hard-to-scale network for several reasons. First, jobs use network to read input data, as the input is randomly spread across several machines in a cluster. Second, jobs usually consist of intermediate data shuffle phases such as shuffle and join. Finally, when data flows generated by consecutive jobs are dependent, network is also used for answering dependent data between jobs. Meanwhile, recent researches have shown that cross-rack network traffic is much higher than the cross-node traffic in clusters. Thus, our goal is to reduce cross-rack network traffic by improving rack-level data locality. To improve the locality, several previous job schedulers employ techniques like delay scheduling to optimize input data locality, but do not address other network-intensive phases. Recent effort Shuffle Watcher attempts to localize.

II. BACKGROUND THEORY

Cloud computing is the on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the user. The term is generally used to describe data centers available to many users over the Internet. Large clouds, predominant today, often have functions distributed over multiple locations from central servers. If the connection to the user is relatively close, it may be designated an edge server. Clouds may be limited to a single organization enterprise clouds, or be available to many organizations public cloud. Cloud computing relies on sharing of resources to achieve coherence and economies of scale. Advocates of public and hybrid clouds note that cloud computing allows companies to avoid or minimize up-front IT infrastructure costs. Proponents also claim that cloud computing allows enterprises to get their applications up and running faster, with improved manageability and less maintenance, and that it enables IT teams to more rapidly adjust resources to meet fluctuating and unpredictable demand. Cloud providers typically use a "pay-as-you-go" model, which can lead to unexpected operating expenses if administrators are not familiarized with cloud-pricing models. The availability of high-capacity networks, low-cost computers and storage devices as well as the widespread adoption of hardware virtualization, service-oriented architecture and autonomic and utility computing has led to growth in cloud computing.[6][7][8] By 2019, Linux was the most widely used operating system, including in Microsoft's offerings and is thus described as dominant. Cloud Service Provider will screen, keep up and gather data about the firewalls, intrusion identification or/and counteractive action frameworks and information stream inside the network.Job scheduling is a critical and challenging task for computer systems since it involves a complex allocation of limited resources such as CPU/GPU, memory and IO among numerous jobs. It is one of the major tasks of the scheduler in a computer system’s Resource Management System (RMS), especially in high-performance computing (HPC) and cloud computing systems, where inefficient job scheduling may result in a significant waste of valuable computing resources. Data centers, including HPC systems and cloud computing systems, have become progressively more complex in their architecture, configuration (e.g., special visualization nodes in a cluster) and the size of work and workloads received, all of which increase the job scheduling complexities sharply.The undoubted importance of job scheduling has fueled interest in the scheduling algorithms on data centers. At present, the fundamental scheduling methodologies, such as FCFS (first-come-first-serve), backfilling, and priority queues that are commonly deployed in data centers are extremely hard and time-consuming to configure, severely compromising system performance, flexibility and usability. To address this problem, several researchers have proposed data-driven machine learning methods that are capable of automatically learning the scheduling policies, thus reducing human interference to a minimum. Specifically, a series of policy based deep reinforcement learning approaches have been proposed to manage CPU and memory for incoming jobs, schedule time-critical workloads, handle jobs with dependency, and schedule data centers with hundreds of nodes.Despite the extensive research into job scheduling, however, the increasing heterogeneity of the data being handled remains a challenge. These difficulties arise from multiple issues. First, policy gradient DRL method based scheduling method suffers from a high variance problem, which can lead to low accuracy when computing the gradient. Second, previous work has relied on used Monte Carlo (MC) method to update the parameters, which involved massive calculations, especially when there are large numbers of jobs in the trajectory.To solve the above-mentioned challenges, we propose a policy-value based deep reinforcement learning scheduling method called A2cScheduler, which can satisfy the heterogeneous requirements from diverse users, improve the space exploration efficiency, and reduce the variance of the policy. A2cScheduler consists of two agents named actor and critic respectively, the actor is responsible for learning the scheduling policy and the critic reduces the estimation error. The approximate value function of the critic is incorporated as a baseline to reduce the variance of the actor, thus reducing the estimation variance considerably. A2cScheduler updates parameters via the multi-step Temporal-difference (TD) method, which speeds up the training process markedly compared to conventional MC method due to the way TD method updates parameters.Job scheduling is the process of allocating system resources to many different tasks by an operating system (OS). The system handles prioritized job queues that are awaiting CPU time and it should determine which job to be taken from which queue and the amount of time to be allocated for the job. This type of scheduling makes sure that all jobs are carried out fairly and on time. Most OSs like Unix, Windows, etc., include standard job-scheduling abilities. A number of programs including database management systems (DBMS), backup, enterprise resource planning (ERP) and business process management (BPM) feature specific job-scheduling capabilities as well. Job scheduling is performed using job schedulers. Job schedulers are programs that enable scheduling and, at times, track computer "batch" jobs, or units of work like the operation of a payroll program. Job schedulers have the ability to start and control jobs automatically by running prepared job-control-language statements or by means of similar communication with a human operator. Generally, the present-day job schedulers include a graphical user interface (GUI) along with a single point of control.

Organizations wishing to automate unrelated IT workload could also use more sophisticated attributes from a job scheduler, for example:

- Real-time scheduling in accordance with external, unforeseen events
- Automated restart and recovery in case of failures
- Notifying the operations personnel
- Generating reports of incidents
- Audit trails meant for regulation compliance purposes

In-house developers can write these advanced capabilities; however, these are usually offered by providers who are experts in systems-management software. In scheduling, many different schemes are used to determine which specific job to run. Some parameters that may be considered are as follows:

- Job priority
- Availability of computing resource
- License key if the job is utilizing a licensed software
- Execution time assigned to the user
- Number of parallel jobs permitted for a user
- Projected execution time
- Elapsed execution time
- Presence of peripheral devices
- Number of cases of prescribed events

Scheduling is a decision-making process that plays an important role in most manufacturing and service industries. It is used in procurement and production, in transportation and distribution, and in information processing and communication.

The scheduling function usually uses mathematical techniques or heuristic methods to allocate limited resources to the processing of tasks. A proper allocation of resources enables the company to optimize its objectives and achieve its goals. Resources may be machines in a workshop, runways at an airport, or crews at a construction set. Tasks may be operations in a workshop, takeoff and landings at an airport, or stages in a construction project. Each task may have a priority level, an earliest possible starting time, and a due date. The objectives may also take many forms, such as minimizing the time to complete all tasks or minimizing the worst performance of the schedule. Many approaches from operations research and artificial intelligence are employed to deal with scheduling problem.

Cloud computing is a popular networking paradigm that provides resources via Internet . Cloud computing helps web service providers reduce hardware infrastructure expenses for deploying their applications. In addition, easy resource management and fast response time are the other interesting characteristics that bring the attentions to the cloud computing the focus is on cloud Infrastructure-as-a Service (IaaS), where infrastructure resources such as network, computing, database, etc. are offered by cloud providers. Cloud providers usually offer two types of IaaS resource provisioning plans, reserved and on demand plans, to web service providers that have different charging schemes based on the resource usage. The reserved plans are often offered for relatively long-term contracts. Using reserved plans, web service providers can get discount rates on reserved resources and pay once for the contract time period (e.g. one-year contract or three-year contract for Amazon EC2). Through on-demand plans, cloud providers offer more flexible resource pricing strategies. On-demand plans charge cloud web service providers on a pay-as yougo basis and enable them to start or terminate instances at any moment according to their needs without paying any penalty. However, comparing the costs of resources per unit of time, on-demand resources are often more expensive than the reserved ones. With the reserved plans, web service providers reserve instances in advance for long-term contracts. Due to ignorance of demand uncertainty in the reserved plans, resource provisioning only with the reserved instances is a challenging task. The purchased resources may not be enough to handle the demands all the time that leads to underprovisioning. This may result in failure in meeting web service providers' Quality of Service (QoS) criteria which is a crucial concern for both cloud providers and web service providers in presence of the uncertainty in the demands [9]. On the other hand, over-provisioning may happen if the allocated resources are excessive to handle actual arrived demands most of the time , leading to unnecessary costs. Recently, some research studies have addressed cloud resource allocation, as optimizing resource provisioning costs has become important for cloud web service providers. Most of the existing approaches in cloud resource allocation, model resource allocation problem as a single phase algorithm. In these works, the authors ignore demand uncertainty by assuming deterministic values for demands. Therefore, the elastic nature of cloud-based applications is not considered. To address the demand uncertainties, in, several dynamic resource allocation algorithms are developed. These algorithms are more flexible and allocate cloud resources dynamically to optimize resource provisioning costs. However, these works do not often exploit the cost benefits of the reserved plans that are offered by the cloud providers. Therefore, they may fail to achieve economical solutions. The propose a hybrid method to allocate cloud resources for deploying cloud-based web applications dynamically. To take advantage of reserved and on-demand resources and achieve a hybrid solution that minimizes total deployment cost and guarantees the QoS under demand uncertainties.

2.1 Major contributions

The Proposing Dynamic Cloud Resource Allocation (DCRA) algorithm that solves the resource provisioning in two reserved and dynamic provision phases, developing a stochastic optimization approach to model the user demands as random variables, and achieving 10% improvement in total deployment costs.

The proposed DCRA algorithm is evaluated using two different benchmark workloads in Amazon Web Services (AWS) [7] and Microsoft Azure cloud [6] as the cloud providers. The results show that the proposed DCRA algorithm finds solutions that minimize total deployment costs under the uncertainties in users' demands.

2.2 Dynamic Cloud Resource Allocation

The existing cloud resource allocation we aim to provide dynamic cloud resource allocation that considers the dual price plan from cloud providers and finds the balance between reserved and on-demand resources. The propose is Dynamic Cloud Resource Allocation (DCRA) algorithm, a two-phase algorithm that minimizes the cost of the web service deployment. The first phase (referred to as the reservation phase), resources from the reserved plan are allocated for web application deployment to meet the minimum QoSrequirements.The second phase (referred to as the dynamic provision phase), non-deterministic user demands are modeled as random variables. A stochastic optimization approach is proposed to dynamically allocate on-demand resources to minimize deployment costs under the on-demand plan subject to QoS requirements.

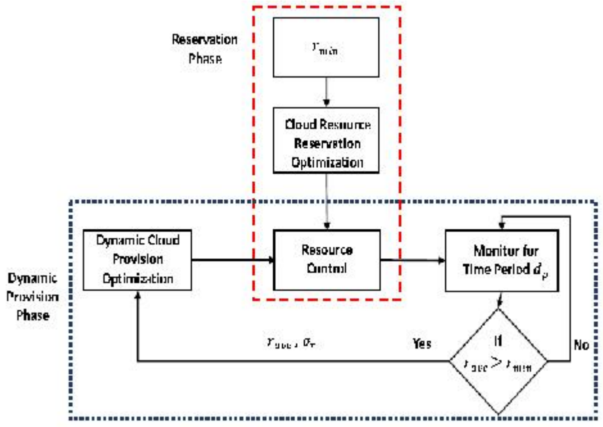


Fig 2.2Cloud Resource Allocation

III. COMPARATIVE STUDY

The Comparative Methodologies:

- The realization of the SA algorithm
- Differential evolution (DE) meta heuristic search algorithm
- Estimation of distribution algorithm(EDA)
- Nash Equilibrium (NE)

3.1The realization of the SA algorithm

Approach

Real-life problems also belong to natural processing -hard issues and have a high dimensional data. Work schedules tasks can be defined as one of the most important optimization problems because plans and schedules need to be organized in all fields. Such tasks, as a rule, are usually modeled as the job-shop scheduling the job problem, which deals with planning multi-stage service systems. However, this view ignores the cyclic nature of the tasks. Since production processes are cycle after-cycle often, it is necessary to make a plan not for a single execution but for multiple ones. In this paper, we deal with a modification of the job-shop scheduling problem for the cyclic production and we substantiate the claim that an optimal solution of the cyclic job-shop task is not limited to the cyclic using a solution of the usual job-shop task. For the purpose of experimental research, the Simulated Annealing (SA) algorithm is used.

Architectural View:

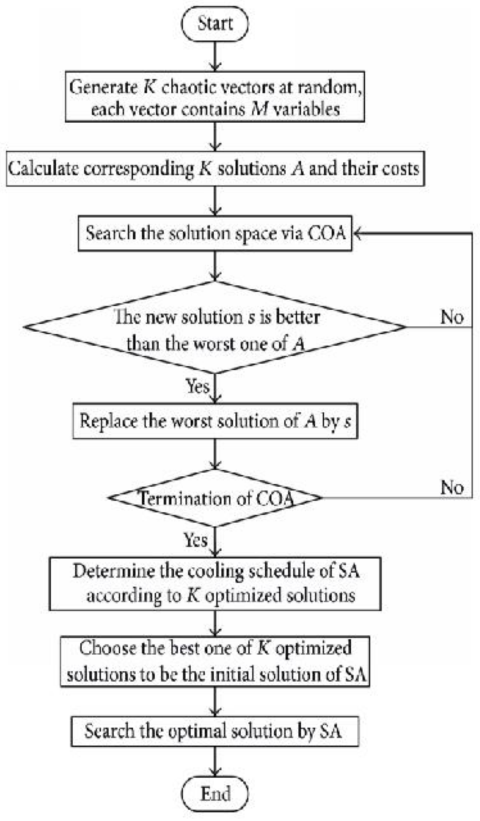

Start
Generate K chaotic vectors at random, each vector contains M variables
Calculate corresponding K solutions A and their costs
Search the solution space via COA
The new solution s is better than the worst one of A
No
Yes
Replace the worst solution of A by s
Termination of COA
No
Yes
Determine the cooling schedule of SA according to K optimized solutions
Choose the best one of K optimized solutions to be the initial solution of SA
Search the optimal solution by SA
End

Fig 3.1 SA algorithm architecture

Algorithm:

```
Start annealing algorithm
{
/* initialization */
temperature ← INITIAL_TEMPERATURE
solution ← initialize()
current_value = schedule_length(solution)
counter_steps ← 0
while (counter_steps< COOLING_STEPS)
temperature ← temperature* COOLING_FRACTION
start_value = current_value
counter_steps_temp ← 0
while(counter_steps_temp< STEPS_TEMP)
 {
/* pick randomly two elements of a schedule to swap
r1 ← random_integer(1, N)
 r2 ← random_integer(1, N)
/*create a new schedule and find it's length */
solution ← swap_schedule(solution, r1, r2)
new_value = schedule_length(solution)
delta = new_value – current_value
if (delta < 0) /*find a better solution*/
{
current_value = new_value
} else /*find a worse solution, use a randomize chose */
{
ex = exp((-delta/current_value)/(KT*temperature))
if (ex >randon_float(0,1)) /*accept new solution */
 {
current_value = new_value
}
else /* reject */
{
solution ← swap_schedule(solution, r2, r1)
}
counter_steps_temp ← counter_steps_temp + 1
}
/* restore temperature if progress has been mad */
if ((current_value - start_value) < 0.0)
{ temperature ← temperature/COOLING_FRACTION;
}
counter_steps ← counter_steps + 1
}
}
End annealing algorithm
```

3.2 Differential evolution (DE) Meta heuristic search algorithm

Approach

Differential evolution (DE) Meta heuristic search algorithm that optimizes a problem by iteratively improving a candidate solution based on an evolutionary process. Such algorithms make few or no assumptions about the underlying optimization problem and can quickly explore very large design spaces. Approach to induce oblique decision trees (DTs) is described. This type of decision trees uses a linear combination of attributes to build oblique hyper planes dividing the instance space. Oblique decision trees are more compact and accurate than the traditional univariate decision trees. On the other hand, as differential evolution (DE) is an efficient evolutionary algorithm (EA) designed to solve optimization problems with real-valued parameters, and since finding an optimal hyperplane is a hard computing task, this meta heuristic (MH) is chosen to conduct an intelligent search of a near-optimal solution. Two methods are described in this chapter: one implementing a recursive partitioning strategy to find the most suitable oblique hyperplane of each internal node of a decision tree, and the other conducting a global search of a near-optimal oblique decision tree. A statistical analysis of the experimental results suggests that these methods show better performance as decision tree induction procedures in comparison with other supervised learning approaches.

Architectural View:

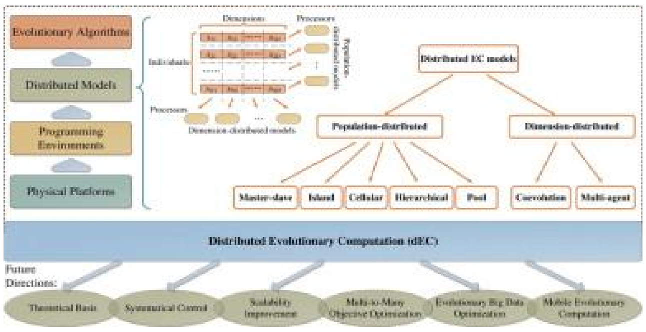
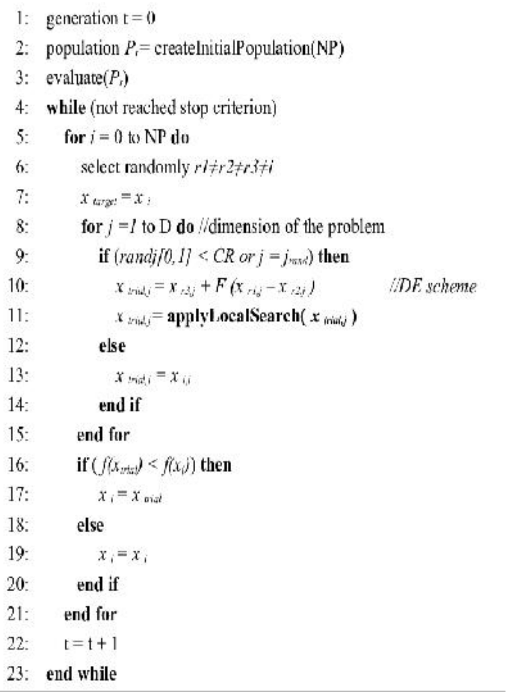


Fig 3.2 Differential evolution (DE) architecture

Algorithm :



```
generation t = 0
population $P_t$ = createInitialPopulation(NP)
evaluate($P_t$)
while (not reached stop criterion)
   for $i$ = 0 to NP do
      select randomly $r1 \neq r2 \neq r3 \neq i$
      $x_{target} = x_i$
      for $j$ = $1$ to D do //dimension of the problem
         if ($randj[0,1] < CR$ or $j = j_{rand}$) then
            $x_{trial,j} = x_{r3,j} + F(x_{r1,j} - x_{r2,j})$          //DE scheme
            $x_{trial,j}$ = applyLocalSearch( $x_{trial,j}$ )
         else
            $x_{trial,j} = x_{i,j}$
         end if
      end for
      if ( $f(x_{trial}) < f(x_i)$ ) then
         $x_i = x_{trial}$
      else
         $x_i = x_i$
      end if
   end for
   t = t + 1
end while
```

3.3 Estimation of distribution algorithm

Approach

An effective estimation of distribution algorithm (EDA) is proposed to solve the flexible job scheduling problem with fuzzy processing time. A probability model is presented to describe the probability distribution of the solution space. A mechanism is provided to update the probability model with the elite individuals. By sampling the probability model, new individuals can be generated among the search region with promising solutions. Moreover, a left-shift scheme is employed for improving schedule solution when idle time exists on the machine. In addition, some fuzzy number operations are used to calculate scheduling objective value. The influence of parameter setting is investigated based on the design of experiment, and a suitable parameter setting is suggested. Numerical testing results and comparisons with some existing algorithms are provided, which demonstrate the effectiveness of the EDA. Estimation of distribution algorithm (EDA) is a relatively new paradigm in the field of evolutionary algorithms, which employs explicit probability distributions in optimization Different from the GA that reproduces a new population with the crossover and mutation operators, the EDA does it implicitly. With the tool of statistical analysis, the EDA tries to estimate the underlying probability distribution of the potential individuals and builds a probability model of the most promising area by statistical information based on the search experience. The probability model is used for sampling to generate the new individuals and is updated in each generation with the elite individuals of the new population. In such an iterative way, the population evolves, and finally satisfactory solutions can be obtained.

Architectural View:

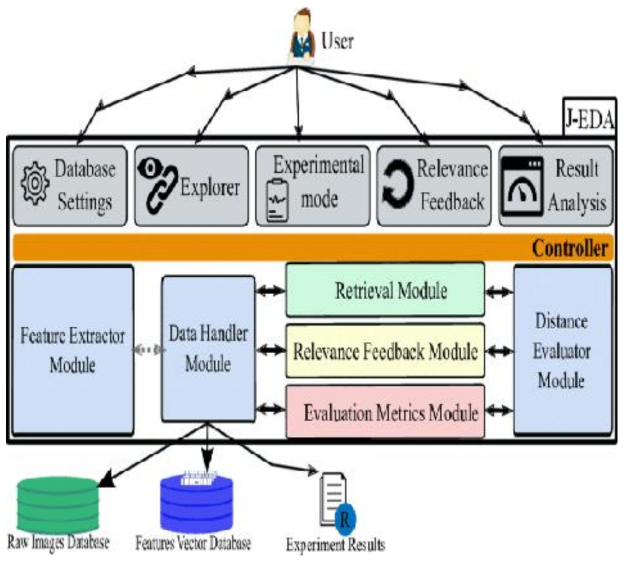


Fig 3.3 EDA architecture

Algorithm:

Step 1: t=0
Step 2: Initialize a probability model p(x,t) (usually a uniform distribution).
Step 3: Generate an infinite sized sample X t from p(x,t) .
Step 4: Evaluate X t in the objective function(s) and constraint(s).
Step 5: Compute the selection distribution p^s (x,t) , and use it as the new search distribution. Step 6: Then: p(x,t+1) = p^s (x,t) . (Selection and model rcomputation step)
Step 7: t=t+1
Step 8: 1If the stop criterion is not reached go to Step 3

4.4 Nash Equilibrium (NE)

Overloaded gateway verifies the estimated QoS index with the prespecified expected value for including only the truthful gateways in the auction process. We, further, establish the existence of Nash Equilibrium (NE) in the proposed scheme. We theoretically analyze the upper and lower pricing limits, and prove that the algorithm reaches optimality. We extend the solution for multiple users' load increment in the presence of multiple misbehaving gateways We summarize the main contributions of this paper as follows:

- We, theoretically, prove the requirement of load sharing in case of increase in demand by a user in MCN environment.
- We design a utility maximization problem for qualityassured secured load sharing (QuaLShare) in MCN with overloaded demand of a gateway.
- We analyze the optimality criteria and the existence of Nash Equilibrium for the proposed scheme, QuaLShare.
- We extend the solution for the multiple users' demand increment problem, followed by its theoretical analysis, for checking the optimality and Nash Equilibrium.

Architectural View:

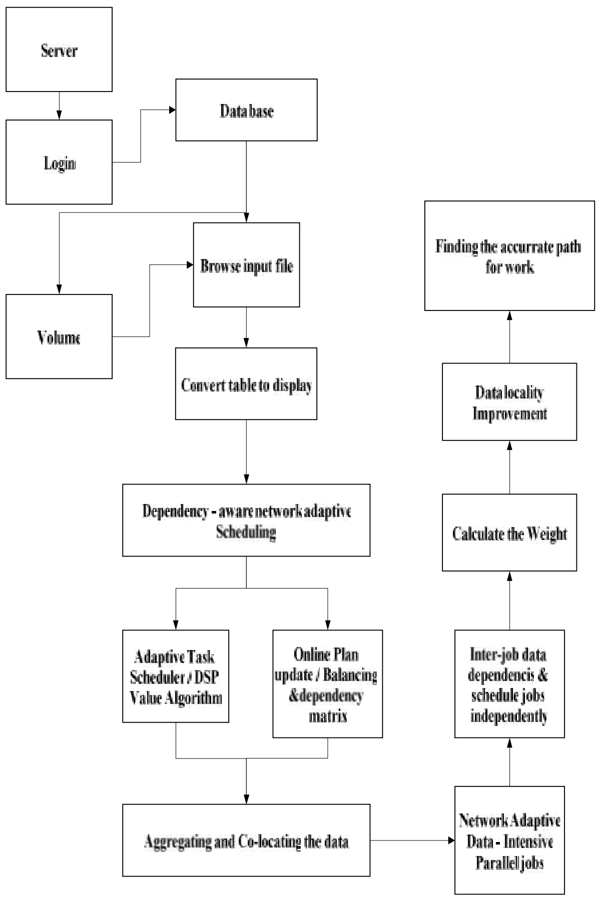

Server
Login
Data base
Browse input file
Volume
Convert table to display
Dependency - aware network adaptive Scheduling
Adaptive Task Scheduler / DSP Value Algorithm
Online Plan update / Balancing &dependency matrix
Aggregating and Co-locating the data
Network Adaptive Data - Intensive Parallel jobs
Inter-job data dependencis & schedule jobs independently
Calculate the Weight
Data locality Improvement
Finding the accurrate path for work

Fig 4.4 Nash Equilibrium architecture

Algorithm:

```
Adaptive task scheduler
Step 1: Receiving task details from all nodes;
Step 2: if A load from the node in rack iindicating free resource then
Step 3: Update online plan;
Step 4: Balance online plan;
Step 5:  if Network Saturated is True then
Step 6: for each job in the parallel jobs do
Step 7: Select the preferred rack with the smallest work load ;
Step8: end for
```

IV. CONCLUSION

In today’s clusters, many real-world workloads inherently exhibit strong inter-job data dependency. However, existing network-aware job schedulers ignore the dependencies and Schedule the jobs independently to one another. Driven by the intuition that aggregating and co-locating the data and tasks of dependent jobs offers an extra opportunity of data locality, we proposed and developed Dawn, a dependency aware network adaptive scheduler for data intensive parallel jobs with dependencies. Dawn is able to determine the optimal racks for each job and adaptively choose the most suitable job to schedule based on the network status. This helps it to maximally exploit rack-level data locality while at the same time smooth network traffic to improve utilization. We implemented Dawn on Apache Yarn. Our experiments show that Dawn can significantly enhance the job performance compared to two state-of-art approaches .

REFERENCES

[1] W. Chen, A. Pi, S. Wang, and X. Zhou, “Characterizing scheduling delay for low-latency data analytics workloads,” in Proc. IEEE Int. Symp. Parallel Distrib. Process., 2018.

[2] The Cyclic Job-Shop Scheduling Problem: The New Subclass Of The Job-Shop Problem And Applying The Simulated Annealing To Solve It: P. V. Matrenin;V. Z. Manusov_2016

[3] A flexible job-shop scheduling for small batch customizing Hu Li;ZheLi; RuiduanYang; HuaweiLu; Youchao Zhang_2016

[4] Job scheduling integrated with imperfect preventive maintenance considering time-varying operating condition: J. Hu;Z. Jiang_2017

[5] Event-driven dynamic job shop scheduling execution based on improved genetic algorithm and ontology: Lingling Xue; PengWang; Haibo Cheng ; Peng Zeng; Haibin Yu_ 2017