Container Cluster Scheduling Strategy Based on Delay Decision Under Multidimensional Constraints

Yijun Xue[1], Ningjiang Chen[1,2(✉)], and Yongsheng Xie[1]

[1] School of Computer and Electronic Information, Guangxi University, Nanning 53004, China
chnj@gxu.edu.cn
[2] Guangxi Key Laboratory of Multimedia Communications and Network Technology, Nanning 53004, China

Abstract. With the rise of online applications such as machine learning, stream processing, and interactive data-intensive applications in shared clusters, container cluster scheduling in data centers is facing new challenges. In order to solve the problem that application performance and economic cost cannot be balanced in a container cluster deploying a hybrid application, this paper proposes a container cluster scheduling strategy based on delay decision under multi-dimensional constraints. Formal language-based application placement constraints were introduced, and a task reorder model was established based on delayed decision-making. The experiments show that this strategy improves application performance and cluster utilization.

Keywords: Container cluster · Multi-dimensional constraint · Delay decision · Application performance · Cluster utilization

1 Introduction

With the extensive applications of Hadoop (processing analysis) [1], TensorFlow (deep learning) [2], core e-commerce [3] and other long-term online running applications, according to the principle that application resource demand is less than supplied physical resource, the application and resource are dynamically adjusted at runtime. It is difficult to apply a cluster resource sharing management model that optimizes supply and demand management.

Modern large cloud data center container clusters usually run many different types of applications. In addition to traditional batch processing applications, they also include stream processing [4], iterative computing [5], data-intensive interactions [6], and delay-sensitive Online application [7]. Studies [8–10] show that in the production environment of actual data center clusters, the operational utilization of global cloud facilities and commercial clusters is only 6% to 12%. We analyzed the set of newly released tracking data [11] by Alibaba. The statistical results are shown in Fig. 1. There are space imbalances (heterogeneous resource utilization across machines) and time imbalances when the cluster is running. (The resource usage time of each machine

J. Zeng et al. (Eds.): ICPCSEE 2020, CCIS 1257, pp. 690–704, 2020.
https://doi.org/10.1007/978-981-15-7981-3_51

varies). Alibaba reserves fixed resources for online applications. Unlike batch jobs for short-term containers, these applications take longer to run, so called long-running application (LRA). Containers of LRA have a relatively long service life, avoiding repeated container initialization costs and reducing scheduling load. The overly simple scheduling strategy will result in poor placement of LRAs tasks, exacerbate these two imbalances in the cluster, and cause waste of resources.

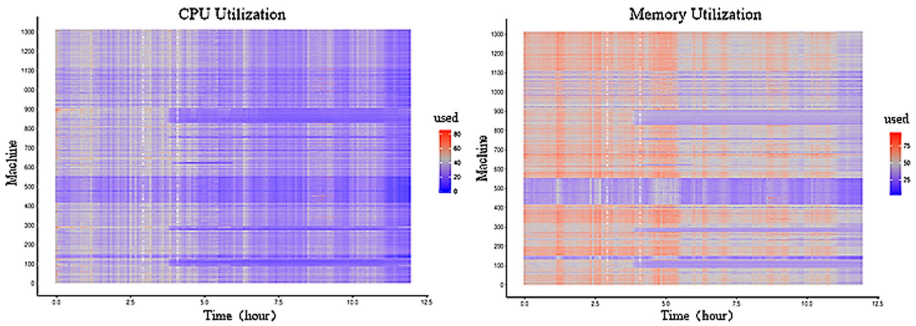


Fig. 1. Current status of cluster resource utilization

This paper proposes a cluster scheduling method based on delayed decision and LRAs. This method uses a formal language to dynamically describe multi-dimensional constraints and construct a constraint management model. At the same time, the optimal task scheduling queue is obtained by using the node matching optimization strategy and multi-attribute decision making. The contributions of this article are mainly three points:

- A multi-dimensional labeling constraint characterization model is proposed, which uses the flexibility of formal language to realize the dynamic expression of multi-dimensional constraints and reduce the constraint violation rate.
- A scheduling management mechanism based on delay decision is proposed, which includes a node matching optimization strategy and a task reorderer. Using the idea of delay, tasks are optimally placed, which improves throughput and data utilization.
- Experimental results show that the average utilization rate of this method is improved by about 10% compared with the existing cluster scheduling methods, and the constraint violation rate does not exceed 5%, which effectively ensures the application performance.

2 Related Work

Some existing scheduling systems for LRA support [12–14] are still in the exploratory stage for the constraints' expression between deployed application containers. Simple affinity and anti-affinity constraints have been partially implemented in a few

scheduling systems, such as Mesos [15], YARN [16], Borg [17], but their constraints are implicitly supported by static machine attributes, lacking some flexibility.

In [16] and [17], YARN and Borg act as matchers between the resource requirements of various applications and the resources available on the machine nodes, but which will cause blocking situations. References [18–20] used a scheduling delay mechanism to alleviate the head-of-line blocking situation caused by FIFO ordering. Apollo [21] and Sparrow [22] independently decide where to run tasks to improve scalability and reduce allocation delays. But for tasks of LRAs, these methods cannot achieve global optimal allocation. Choosy [23], a resource fair scheduling method, realizes resource sharing under placement constraints, but lacks research on better convergence time and constraints between tasks. Paper [24] proposed the Quincy scheduling strategy to instantly calculate and optimize the global matching of scheduling decisions through the minimum flow graph. Due to the high complexity of the graph, there will be a huge delay in the cluster scheduling of large-scale data centers. To sum up, the existing methods are difficult to achieve flexible expression of multi-dimensional constraints, and cannot balance scheduling resources and application performance. There is a high constraint violation rate and a low resource utilization rate.

3 General Framework of the Method

The approach overview is shown in Fig. 2. In this paper, constructing a constraint management model with multi-dimensional constraints based on the formal language of each application. The node matching optimizer and task reorder are used to determine the scheduling order of the tasks, so as to achieve the optimal placement of tasks.

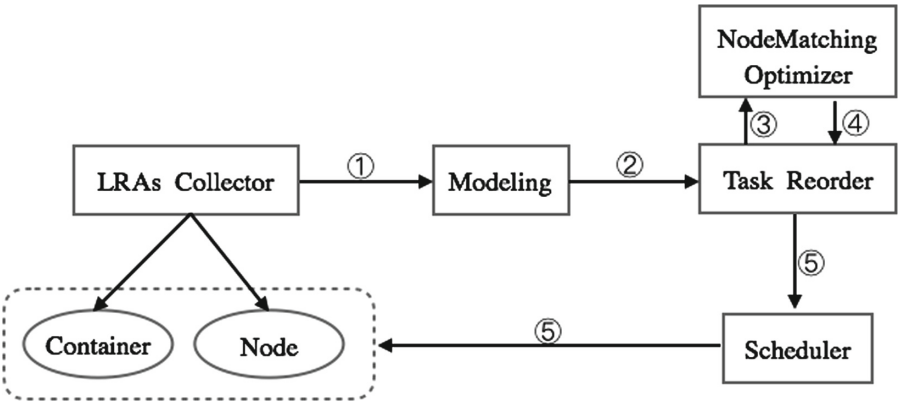


Fig. 2. Approach overview (The flow of the system approach can be described as follows)

① The LRAs collector collects the load and CPU, memory and other resource usage of each container and node.
② A performance modeler that builds a constraint management model and characterizes the constraint relationships between containers and nodes.
③ The Task Reorder analyzes multiple characteristics of tasks through multi-attribute decision-making, establishes a decision matrix, obtains the optimal task queue scheduling order, and achieves optimal task placement.
④ The node matching optimizer uses the resource utilization rate in the step ① as a benchmark, uses a set of priority functions to process the nodes in the cluster, and passes the processing results to the task reorder in real time.
⑤ The container scheduler performs a certain delay scheduling according to the task queue scheduling order and node processing results.
⑥ After performing container scheduling, continue to execute the steps to form a method closed loop.

4 Dynamic Characterization of Multidimensional Label Constraints

In the actual production environment, cluster scheduling needs to meet various constraints, but there will always be conflicts, so it is necessary to ensure a low constraint violation rate. It has an important impact on the performance of the application, which is possible to further optimize the core of the cluster. Label is a simple but powerful constraint mechanism for referencing containers of the same or different (possibly not yet deployed) applications. For example, using the label 'hb' to reference the current and future containers of HBase application. This paper uses a formal language (label) to specify multi-dimensional constraints for long-running application containers, build a constraint management model, as shown in Fig. 3.

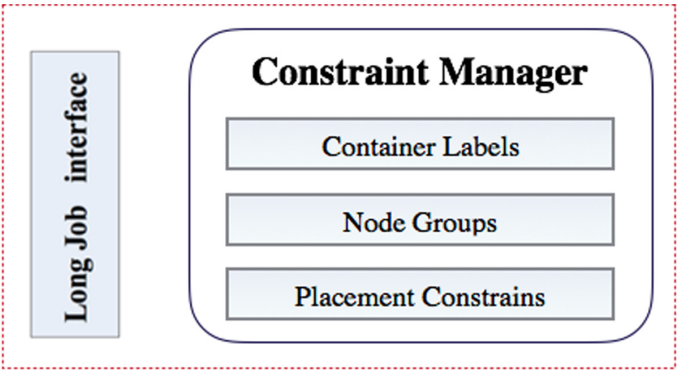


Fig. 3. Constrains Manager Model, and simply record container information generated by the application, including application ID, application type, deployment node, and resource specifications. The node label set is updated in real time. When a container is assigned to a node, the container label is added to the node label set. Only when the container has completed execution and is not in a running state, the label associated with the node is removed.

(1) Establish the set of tags: Setting the label set in the unit of node, each label can be associated with multiple containers on the node;
(2) Specify node group: The predefined node group is a node and a rack. A node corresponds to a single element of a cluster node. The rack contains all nodes of a physical machine.
(3) Define constraint forms: Allow application owners and cluster operators to use labels to specify container placement constraints that reference containers of the same or different applications, and point to a specific node set of node groups. The form of the constraint defined in this paper is as follows:

$$P = \{\text{subject\_label, label\_constraint, node\_group}\}$$

subject_label is a label or label association that identifies the container subjected to constraints; ***label_constraint*** is a constraint of the form {p_label, p*min*, p*max*}, p_label is the container label (or label association), p*min* and p*max* are positive integers and represent the number of containers; ***node_group*** represents a node group.

At this paper, there are the following constraints:

Definition 1: Affinity constraints. Coordinating some LRA containers on the same node or node group can bring benefits to the cluster. Use p*min* = 1 and p*max* = ∞ to express affinity constraints.

Definition 2: Anti-affinity constraint. Minimize resource interference between long-running applications by placing containers on different machines through anti-affinity constraints inside and between applications, and use p*min* = 0 and p*max* = 0 to express anti-affinity (anti-affinity) constraints.

Definition 3: Cardinality constraint. Affinity and anti-affinity constraints represent the two extremes of container placement. In order to achieve a balance between the two, a more flexible cardinality constraint is used, which limits the number of juxtaposed containers. Cardinality constraints are expressed for other values of p*min* and p*max*.

5 Scheduling Management Mechanism Based on Delayed Decision

Considering the constraints between tasks, a reasonable placement decision can be made, but in the implementation process, it was found that the placement order of the tasks also has an important impact on resource utilization and constraint violation rates. As shown in Fig. 4. (a), there are idle resources in the cluster, but there are still waiting task queues, which reduces the utilization of cluster resources and affects the running time of jobs, resulting in poor quality of service.

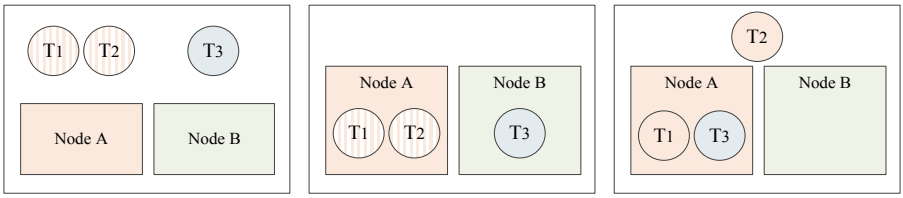


Fig. 4. Task deployment results with different queue order: it is assumed that there are two tasks (T1, T2) have affinity constraints and can only be deployed on node A; task T3 is unconstrained; each node can only deploy two tasks. The ideal deployment result is Fig. 4. (b), and the actual deployment result is Fig. 4. (c).

Compared with the queue management functions of existing scheduling systems, most of them support queuing on nodes, but do not support global task queuing. This paper combines the multi-dimensional constraints between tasks and considers multiple task requests at once, and proposes a scheduling management mechanism based on delay decision, including node matching optimization strategy and task queue reordering model.

5.1 Strategy of Node Matching Optimization

In practical application scenarios, it is often the case that the working nodes to be scheduled are "picked", that is, certain containers are required to be scheduled to run only on specific hosts, so a node matching optimization strategy is designed in this paper. The nodes are matched to achieve container scheduling to eligible nodes, and the nodes are preferably used to implement container scheduling to appropriate nodes.

Considering multiple task requests within a certain time interval π, define a task set $T = \{T1, T2, \ldots, Tn\}$. For each task Ti, the basic filtering generally evaluates some common factors such as node port availability, whether resources are satisfied, and whether the mounted disks conflict. The constraint filtering generally considers the anti-affinity constraint relationship between tasks, and takes the difference set. Find the node set Ni matched by each task.

Use a set of priority functions to process each node in the node set Ni, AffinityPriority, AntiAffinityPriority (there is a constraint relationship between Ti and the task running in node Nij. Return a function value of 1, otherwise Returns 0). This article uses a data collector to obtain the total CPU, memory (Nodeij.capacityCPU, Nodeij.capacityMemory) on the node, and the sum of the requested CPU and memory (Nodeij.requestCPU) of the container that has been scheduled on this node and the Ti to be scheduled, Nodeij.requestMemory). The specific formula is as follows:

$$Node_{ij}.restCPU = \left(\frac{Node_{ij}.capacityCPU - Node_{ij}.requestCPU}{Node_{ij}.capacityCPU}\right) \quad (1)$$

$$Node_{ij}.restMemory = \left(\frac{Node_{ij}.capacityMemory - Node_{ij}.requestMemory}{Node_{ij}.capacityMemory}\right) \quad (2)$$

$$priorityFunc4 = \frac{1}{2}(Node_{ij}.restCPU + Node_{ij}.restMemory) * 10 \tag{3}$$

$$priorityFunc5 = 10 - \left| \left(\frac{Node_{ij}.requestCPU}{Node_{ij}.capacityCPU} \right) - \left(\frac{Node_{ij}.requestMemory}{Node_{ij}.capacityMemory} \right) \right| * 10 \tag{4}$$

Among them, Nodeij.restCPU and Nodeij.restMemory represent the remaining rates of CPU and memory resources in Nodeij respectively; the priority function priorityFunc4 is used to evaluate the resource consumption of the node; the priority function priorityFunc5 is used to evaluate the resource balance of the node. The node's final score (anti-affinity constraint) is obtained by adding the values returned by multiple priority functions. The larger the score, the better the quality of the node. According to the order of FianlScoreNodeij, the Ti queue of task Ti candidate nodes is formed.

$$FianlScoreNode_{ij} = \sum_{t}^{m} wt * priorityFunct \tag{5}$$

5.2 Task Priority Multi-attribute Ranking Model

Tasks have multiple characteristics, which lead to different scheduling orders affecting the optimal placement of tasks. This paper analyzes the multiple characteristics of tasks and introduces multi-attribute decision theory to transform the reordering of task queues into multi-attribute decision problems. The number of nodes matched by the task, the amount of requested resources of the task, and the starvation status of the task are of great significance to the task, and their corresponding attribute values are all expressed in the form of real numbers. This article first makes relevant definitions and assumptions, and then builds a Task Reorder Model (**TRM**).

Definition 4: Starvation time STi (Starved Time) of task Ti.

$$\mathrm{STi} = (\mathrm{t} - \mathrm{ai}) - \pi \tag{6}$$

STi represents the length of the starvation time of the task Ti, which reflects the starvation state of the task. It is calculated and determined by the current time t, the task's arrival time ai, and the scheduling interval π, and they have the same unit of measurement.

Definition 5: Number of Matched Nodes (NMNi) for task Ti.

$$\mathrm{NMNi} = \mathrm{NiQueue\_Size} \tag{7}$$

NMNi represents the number of matching nodes of task Ti, and the queue length of the queue NiQueue of task Ti candidate nodes.

Definition 6: Requested Resource (RRi) of task Ti.

$$\mathrm{RRi} = \mathrm{Resquest.CPU} + \mathrm{Resquest.Memory} \tag{8}$$

RRi represents the sum of the total request amount of CPU resources and memory resources of task Ti. Under certain node resources, deploying more tasks can improve the system throughput.

Multi-attribute decision-making needs to evaluate any task Ti from different attributes to get m evaluation results, which corresponds to the attribute vector of the task Ti (Ai1, Ai2, Ai3, …, Aim). Collect the task set T according to the evaluation attribute set Attr Attributes of each task in Ti, thus establishing a decision matrix. The existing evaluation attributes include the three characteristics of task starvation time STi, the number of matching nodes NMNi, and the resource request amount RRi. Therefore, the decision matrix corresponding to this article is shown in the following formula.

$$E = \begin{pmatrix} ST_1 & NMN_1 & RR_1 \\ ST_2 & NMN_2 & RR_2 \\ \vdots & \vdots & \vdots \\ ST_n & NMN_n & RR_n \end{pmatrix} \tag{9}$$

This paper proposes a task queue reordering model (TRM) based on multi-attribute decision making, which is expressed as **{Attr, T, ρ, E, β, w, ψ, RANK}**. Each element in the model is:

Attr is the task evaluation attribute set. In this paper, Attr = {starvation time (STi), number of matching nodes (NMNi), resource request amount (RRi)}; **T** is the task set that arrives within the scheduling interval π, T = {T1, T2, …, Tn}; **ρ** represents the mapping relationship: Ti → (Ai1, Ati2, Ai3, …, Aim), which represents the evaluation of multi-dimensional features of task Ti according to the task evaluation attribute set Attr; **E** means according to the task set Decision matrix $E = \begin{pmatrix} E_{11} & \cdots & E_{1m} \\ \vdots & \ddots & \vdots \\ E_{n1} & \cdots & E_{nm} \end{pmatrix}$ based on attributes of each task,among them, i in Eij represents the i-th task in the task set T, and j represents the j-th task evaluation attribute value of the i-th task, i {1, 2, …, n}, j {1, 2, …, m}; **β** indicates the mapping relationship Attr → w, and the subjective setting of the weight vector w according to the evaluation attributes; **w** represents the task attribute evaluation attribute weight vector, w = (w1, w2, …, wn) T and the matrix E in the task priority decision process Decide the results together. **ψ** represents the mapping relationship (w, E) → RESULT of the decision process. Task priority selection is performed based on the attribute evaluation attribute weight vector w and decision matrix E, and the ranking result of the T task set is calculated. RESULT represents the ranking result of the task. **RANK** = {r1, r2, …, rn}, where rk = {Tk, rNbk}, k {1, 2, …, n}.

Using this model to analyze task evaluation attributes, starvation time (STi) and resource request volume (RRi) are all benefit attributes. The number of matching nodes (NMNi) is a cost attribute. Because the above three task attribute values have different

dimensions, in order to eliminate the influence on the ranking decision result, these task attribute values are processed as follows respectively.

The normalization processing of task evaluation attribute starvation time (STi) and resource request amount (RRi) is as follows:

$$rij = \frac{aij - \min\limits_{i} aij}{\max\limits_{i} aij - \min\limits_{i} aij} \tag{10}$$

The normalized processing of the task evaluation attribute matching node number (NMNi) is as follows:

$$rij = \frac{\max\limits_{i} aij - aij}{\max\limits_{i} aij - \min\limits_{i} aij} \tag{11}$$

$\max\limits_{i} aij$ and $\min\limits_{i} aij$ represent the maximum and minimum values of the element aij in the j-th feature attribute Aj in the i-th task, rij ∈ [0, 1]. The processed decision matrix is D = (rij) n * m, and then a weighted normalization decision matrix C = wD is reconstructed by considering the weight w.

$$C = \begin{pmatrix} w_1 r_{11} & w_2 r_{12} & \cdots & w_m r_{1m} \\ w_1 r_{21} & w_2 r_{22} & \cdots & w_m r_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ w_1 r_{n1} & w_2 r_{n2} & \cdots & w_m r_{nm} \end{pmatrix} \tag{12}$$

Finally, the priority multi-attribute decision evaluation value of task Ti can be expressed as:

$$C(Ti) = \sum_{j=1}^{m} w_j r_{ij}, i = 1, 2, \ldots, n \tag{13}$$

Each task in the task set is sorted according to the value of C (Ti), and the reordering result RANK is obtained, and finally an optimal task scheduling queue is formed. The TRM algorithm is shown below.

Algorithm: Task queue reordering algorithm

Input：T，The set of tasks that arrive within the scheduling interval π；

N_iQueue，Candidate node queue of T_i；

w, The set of task attribute weight

Output：*TQueue*，Task queue

1 T ={ T1,T2,…,T*n* }，N_iQueue = ∅

2 **while** (each $T_i \in T$)

3 $STi = (t - ai) - \pi$

4 $NMNi = N_iQueue\_Size$

5 $RRi = Resquest.CPU + Resquest.Memory$

6 *E=build_matrix(STi, NMNi, RRi)*

7 **for**($i = 1, i \le n, i++$)

8 **if**($j < m$)**then**

9 r_{ij}=function_Normalized(*STi, NMNi, RRi*)

10 **end if**

11 D= $build\_matrix(r_{ij})$

12 C= $build\_matrix(w, r_{ij})$

13 $$C(T_i) = \sum_{j=1}^{m} w_j r_{ij}$$

14 *Rank*=get_set(C(T_i))

15 *TQueue=bulid_queue(RANK)*

16 **end**

6 Experiment

The implementation of the cluster scheduling prototype system MD-Kubernetes designed in this paper is based on Kubernete. Kubernete is a relatively comprehensive container scheduling system so far. The overall design is shown in Fig. 5, which mainly includes Client, Kube-API Server, Resource Manager, Gateway, Node Manager and other modules.

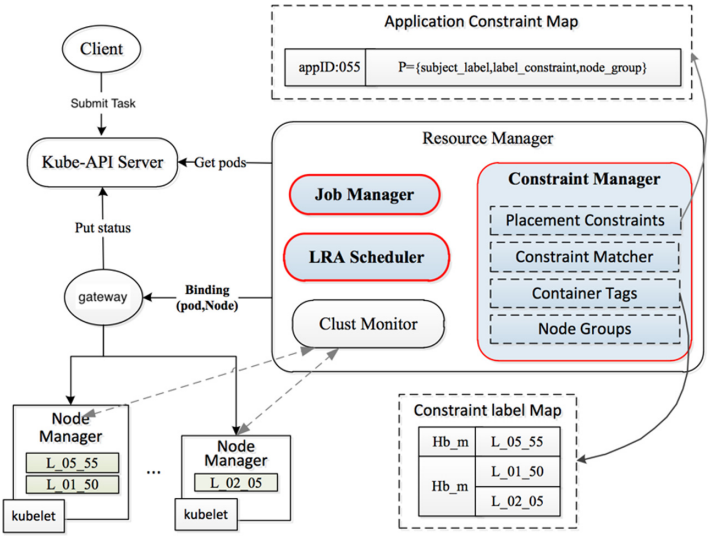


Fig. 5. Architecture of the Prototypal System

The experiment is compared with the existing scheduling system to verify the effectiveness of the system model in the container cluster scenario of hybrid applications. In the experiment, 10 blade server machines were selected to build a 400 virtual node cluster simulation system, configured with Intel Core i7, CPU3.40 GHz, 8 GB memory and Gigabit network card, and these machines were divided into 10 racks.

Software Configuration: In order to realize various configurations in the experiment, this article extends the workload generator GridMix, which can generate long-term running application examples of custom constraints. This article mainly deployed the following applications in the experimental cluster:

(1) HBase instance. Each instance is set up with 10 workers, including simple operations such as adding, deleting, modifying, and checking. Through the API, benchmark tests are performed using YCSB and 0.5 TB data.
(2) TensorFlow instance. Each instance is set up with 8 workers and 2 parameter servers, and runs a machine learning workload involving more than 1,200 iterations.

Each worker is set up with the corresponding configuration, where HBase and TensorFlow workers use containers configured with <2 GB, 1 CPU>, and the main workers of TensorFlow instances use containers configured with <4 GB, 1 CPU>.

Placement Constraints: When deploying HBase and TensorFlow instances, we use the following placement constraints:

(1) Set **affinity constraints** within the application to minimize network traffic. All workers deploying the same HBase instance or TensorFlow instance should be on the same rack;

(2) **Anti-affinity constraints** for different applications. The containers generated by setting HBase instances and TensorFlow instances should be deployed in different racks, thereby improving the stability of the service itself.
(3) This paper implements the **cardinality constraints** between applications. No more than two HBase workers or four TensorFlow workers are placed on the same node, which can minimize resource interference.

This article uses the following three scheduling systems for comparison:

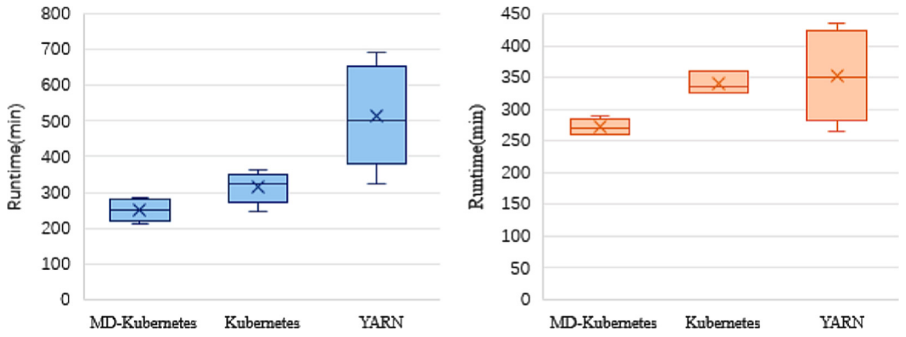
MD-Kubernetes: This paper expresses constraints dynamically. Considering multiple task requests at a time, there is a certain delay. Compared with long-term running applications, the running time is negligible.

YARN: only supports the affinity for specific nodes/racks, and lacks support for constraints between applications. Through comparison, you can intuitively see the impact of constraints on task placement.

Kubernetes: The Kubernetes scheduling system is by far the most complete system that supports placement constraints, but Kubernetes considers one container request at a time during scheduling and does not support cardinality constraints.

6.1 Comparison of Application Performance

Deploy 45 TensorFlow instances and 50 HBase instances in the above 400-node cluster, and submit a job request that uses 50% of the cluster's memory through the GridMix workload generator. The running time of the application indicates different application performance. Figure 6. (a) describes the running time of the deployed machine learning workflow on TensorFlow, and Fig. 6. (b) describes the running time of data insertion on the Hbase instance. Box plots to show their runtimes. It can be seen from the comparison that compared with YARN that does not support constraints between tasks, the running time of the MD-Kubernetes scheduling instance is reduced by 2.1 times and 2.4 times from the median and maximum values of YARN. Unpredictability, because the satisfaction of some constraints is random. Compared with Kubernetes with simple constraints, the running time of TensorFlow instances scheduled by MD-Kubernetes is reduced by 32% at the median, and the running time of HBase instances is reduced by 23% at the median.



(a) TensorFlow (b) HBase insert

Fig. 6. Comparison of application performance

6.2 Comparison of Resource Utilization

In order to simulate the real generation of the cluster, GridMix was used to generate the same load in a 400 virtual node cluster simulation system, and the CPU and memory resource usage of the cluster was continuously monitored to calculate the average CPU and memory utilization efficiency of all open containers. By analyzing the average CPU and memory usage efficiency of a container that is open, you can measure the proportion of pods (containers) that are open but not working (that is, the idle rate of container use). When using Kubernetes for scheduling in a cluster, the average CPU resource utilization of a node is mostly between 20% and 40%, and the average resource utilization of a node's memory is mostly between 40% and 60%. Pass this verification MD-Kubernetes system has the advantage of high resource utilization in terms of task placement strategy (Fig. 7).

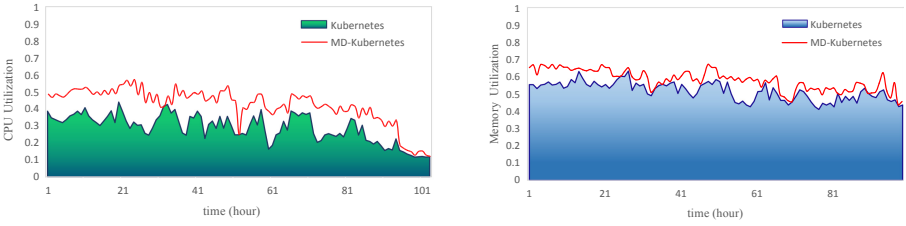


Fig. 7. Average resource utilization of cluster nodes

7 Conclusions

This paper proposes a constraint expression model based on label to support the dynamic expression of multi-dimensional constraints for long-term running tasks in a cluster. A cluster-oriented delay decision scheduling management mechanism guarantees the placement quality of long-term running tasks under multi-dimensional constraints and improves cluster efficiency. The designed MD-Kubernetes prototype system is implemented on Kubernetes in the form of plug-ins. The constraint expression model in this paper also has limitations, and further research on the adaptiveness of cluster scheduling is needed.

Acknowledgment. This work is supported by the Natural Science Foundation of China (No. 61762008), the Guangxi Natural Science Foundation Project (No. 2017GXNSFAA198141), and the National Key Research and Development Project of China (No. 2018YFB1404404).

References

1. Vavilapalli, V.K., Murthy, A.C., Douglas, C.: Apache Hadoop YARN: yet another resource negotiator. In: Symposium on Cloud Computing, pp. 1–16. ACM (2013)
2. Martín, A.: TensorFlow: learning functions at scale. In: ACM Sigplan International Conference on Functional Programming, p. 1. ACM (2016)
3. Verma, A., Pedrosa, L., Korupolu, M.: Large-scale cluster management at Google with Borg. In: Tenth European Conference on Computer Systems, pp. 1–17. ACM (2015)
4. Xingcan, C., Xiaohui, Y., Yang, L.: Overview of distributed stream processing technology. Comput. Res. Develop. **52**(2), 318–332 (2015)
5. Abadi, M., Barham, P., Chen, J.: TensorFlow: a system for large-scale machine learning. In: Usenix Conference on Operating Systems Design and Implementation, pp. 265–283. USENIX Association (2016)
6. Zaharia, M., Chowdhury, M., Franklin, M.J.: Spark: cluster computing with working sets. In: Usenix Conference on Hot Topics in Cloud Computing, p. 10. USENIX Association (2010)
7. Apache HBase[EB/OL] (2018). http://hbase.apache.org
8. Jyothi, S.A., Curino, C., Menache, I.: Morpheus: towards automated SLOs for enterprise clusters. In: Usenix Conference on Operating Systems Design and Implementation, pp. 117–134. USENIX Association (2016)
9. Rajan, K., Kakadia, D., Curino, C.: PerfOrator: eloquent performance models for Resource Optimization. In: ACM Symposium on Cloud Computing, pp. 415–427. ACM (2016)
10. Xu, G., Xu, C.-Z.: Prometheus: online estimation of optimal memory demands for workers in in-memory distributed computation. In: ACM Symposium on Cloud Computing, pp. 655–667. ACM (2017)
11. Alibaba trace [DB/OL] (2018). https://github.com/alibaba/clusterdata
12. Nathuji, R., Kansal, A., Ghaffarkhah, A.: Q-clouds: managing performance interference effects for QoS-aware clouds. In: European Conference on Computer Systems, Proceedings of the, European Conference on Computer Systems, EUROSYS 2010, Paris, France, April, pp. 237–250. DBLP (2010)
13. Avadi, B., Abawajy, J., Buyya, R.: Failure-aware resource provisioning for hybrid Cloud infrastructure. J. Parallel Distrib. Comput. **72**(10), 1318–1331 (2012)

14. Tumanov, A., Zhu, T., Park, J.W.: TetriSched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In: Eleventh European Conference on Computer Systems, pp. 35–36. ACM (2016)
15. Hindman, B., Konwinski, A., Zaharia, M.: Mesos: a platform for fine-grained resource sharing in the data center. In: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, pp. 429–483. USENIX Association (2010)
16. Karanasos, K., Suresh, A., Douglas, C.: Advancements in YARN resource manager **43**(3), 51–60 (2018)
17. Verma, A., Pedrosa, L., Korupolu, M.: Large-scale cluster management at Google with Borg. In: Tenth European Conference on Computer Systems, pp. 1–17. ACM (2015)
18. Ananthanarayanan, G., Kandula, S., Greenberg, A.: Reining in the outliers in map-reduce clusters using Mantri. In: Usenix Conference on Operating Systems Design and Implementation, pp. 265–278. USENIX Association (2010)
19. Ferguson, A.D., Bodik, P., Kandula, S.: Jockey: guaranteed job latency in data parallel clusters. In: European Conference on Computer Systems, EUROSYS, pp. 99–112 (2012)
20. Zaharia, M., Konwinski, A., Joseph, A.D.: Improving MapReduce performance in heterogeneous environments. In: Usenix Conference on Operating Systems Design and Implementation, pp. 29–42. USENIX Association (2008)
21. Boutin, E., Ekanayake, J., Lin, W.: Apollo: scalable and coordinated scheduling for cloud-scale computing. In: Usenix Conference on Operating Systems Design and Implementation, pp. 285–300. USENIX Association (2014)
22. Ousterhout, K., Wendell, P., Zaharia, M.: Sparrow: distributed, low latency scheduling. In: Twenty-Fourth ACM Symposium on Operating Systems Principles, pp. 69–84 (2013)
23. Ghodsi, A., Zaharia, M., Shenker, S.: Choosy: max-min fair sharing for datacenter jobs with constraints. In: ACM European Conference on Computer Systems, pp. 365–378 (2013)
24. Isard, M., Prabhakaran, V., Currey, J.: Quincy: fair scheduling for distributed computing clusters. In: IEEE International Conference on Recent Trends in Information Systems, pp. 261–276 (2009)