Perph: A Workload Co-location Agent with Online Performance Prediction and Resource Inference

Jianyong Zhu[1†], Renyu Yang[2†], Chunming Hu[1*], Tianyu Wo[1], Shiqing Xue[1], Jin Ouyang[3], Jie Xu[21]
[1]BDBC, Beihang University [2]University of Leeds [3]Alibaba Group
{*zhujy, hucm, woty, xuesq* }*@act.buaa.edu.cn;* {*r.yang1, j.xu*}*@leeds.ac.uk; jin.oyj@alibaba-inc.com*

***Abstract*—Striking a balance between improved cluster utilization and guaranteed application QoS is a long-standing research problem in cluster resource management. The majority of current solutions require a large number of sandboxed experimentation for different workload combinations and leverage them to predict possible interference for incoming workloads. This results in non-negligible time complexity that severely restricts its applicability to complex workload co-locations. The nature of pure offline profiling may also lead to model aging problem that drastically degrades the model precision. In this paper, we present Perph, a runtime agent on a per node basis, which decouples ML-based performance prediction and resource inference from centralized scheduler. We exploit the sensitivity of long-running applications to multi-resources for establishing a relationship between resource allocation and consequential performance. We use Online Gradient Boost Regression Tree (OGBRT) to enable the continuous model evolution. Once performance degradation is detected, resource inference is conducted to work out a proper slice of resources that will be reallocated to recover the target performance. The integration with Node Manager (NM) of Apache YARN shows that the throughput of Kafka data-streaming application is 2.0x and 1.82x times that of isolation execution schemes in native YARN and pure cgroup cpu subsystem. In TPC-C benchmarking, the throughput can also be improved by 35% and 23% respectively against YARN native and cgroup cpu subsystem.**

***Index Terms*—performance isolation, co-location, multi-dimensional resource**

I. Introduction

Purchasing commodity servers usually accounts for 50% to 70% of the total cost of Cloud vendors and service providers [1]. However, data center utilization is only between 10% to 50% [2][3]. Cluster administrators are facing great pressure to improve cluster utilization through workload co-location [4][5]. Long running applications (LRA) such as transactional and analytical workloads share the same resource with batch-mode data processing jobs by either time multiplexing [6] or fair sharing according to fixed or dynamic quota on a node basis [7][8]. Guaranteeing performance of LRAs, however, is far from settled as unpredictable interference across applications is catastrophic to QoS [9]. In reality, QoS violation still frequently manifests.

Interference can be mitigated or avoided through performance prediction and performance isolation. Current solutions such as [10][11][12][13] usually employ a large number of sandboxed and offline profiling for different workload combinations and leverage them to predict incoming interference. However, the time complexity and strong assumption of known resource requirement limit the applicability to complex co-locations. This situation usually results in a strict dependence on centralized architecture in which the prediction model is generated in advanced and leveraged in the following decision making. However, offline-based centralized approaches have to encounter scalability and *model aging* problem. Furthermore, multi-resource dimensions (e.g., LLC contention) that are not completely included by existing works but have impact on performance interference need to be considered [14].

Hence, these issues entail a new framework to harness runtime performance and mitigate the involved time cost with continuous and adaptive machine intelligence. It is desirable to explore a quantitative relationship between allocated resource and consequent workload performance, not relying on analyzing interference derived from different workload combinations. Workload co-location also necessitates fine-grained isolation and access control mechanism. Once performance degradation is detected, dynamic resource adjustment will be enforced and application will be assigned an access to specific slices of each resources. Inferring a *just enough* amount of resource adjustment ensures the application performance can be secured whilst improving co-location efficiency and system utilization.

This paper describes Perph, a decentralized framework on a per node basis that decouples ML-based performance prediction from the central resource scheduler. Assuming strong resource isolation can be enforced, Perph agent exploit the sensitivity of long-running applications to multi-resources for quantitatively establishing a relationship between resource allocation and consequential performance. It encompasses multi-dimension resources such as cores, caches, main memory and memory bandwidth, etc. that can overcome the inaccuracy of single dimension based approaches. To deal with model aging, we use Online Gradient Boost Regression Tree (OGBRT) to warmly start from offline training based on a small sampling volume but continuously evolve with model parameters updated. Once performance degradation is detected, resource inference is conducted to work out a proper slice of resources that will be reallocated to recover the target performance. We adopt Intel Resource Director Technology (RDT) [15] for measuring and manipulating memory bandwidth and last level cache utilization/misses. Our prototype is integrated with Node Manager of Apache YARN and we mainly use transactional and analytical application and batch jobs to represent real-world workloads to validate the proposed mechanisms. Ex-

†: co-first authors with equal contribution. ∗: corresponding author.

periments show that the throughput of Kafka data-streaming application is 2.0x and 1.82x times that of isolation execution schemes in native YARN and pure cgroup cpu subsystem. In TPC-C benchmarking, the throughput can also be improved by 35% and 23% respectively against YARN native and cgroup cpu subsystem. In fact, Perph can be applied into any resource management systems and facilitate node daemon to harness application's performance. Particularly, main contributions are as follows:

- An agent that decouples ML-based performance prediction and resource inference from centralized scheduler.
- An online performance model that warmly starts with offline profiling and training to depict multi-dimensional resources and pertaining performance but is continuously updated exploiting incoming workloads.
- An adaptive resource reallocation mechanism based on timely resource inference and multi-resource isolated execution to ensure application performance.

Organization. Section II states problems and architecture overview. Section III and IV detail online performance prediction model and the runtime access control. We detail the system implementation in Section V and experiment evaluation in Section VI. Following the related work in Section VII, we draw conclusions and outline further work.

II. Perph Overview

A. Background and Requirements

Cloud data centers are confronted with a dilemma between application performance and cluster resource utilization. The contention on shared resources is prone to severe performance interference, which is detrimental to QoS targets. In this scenario, how to ascertain a predictable performance model and safe workload co-location is very critical to service provisioning, specifically for LRAs. There are two urgent requirements: **[R1]** It is desirable to explore a quantitative relationship between allocated multi-resources and consequent workload performance, not relying on analyzing interference derived from different workload combinations. Allocation and contention of different resources among applications may have strong impact on application performance. For instance, LLC contention has been illustrated the main source of performance degradation of LRAs [9][16][17]. The size of dataset used in latency-sensitive applications is usually larger than cache capacity. The application's demand for MBW will increase with the increase of system load. Reversely, we can also use the model to infer different resource plans that can achieve a particular performance level. Namely, we take current resource allocation, system loads and target performance as inputs and yield a new resource plan that can generate a specific performance rescue. **[R2]** Workload co-location also necessitates fine-grained isolation and access control mechanism. Once performance degradation is detected, dynamic resource adjustment will be enforced and application will be assigned an access to specific slices of each resources. Inferring a *just enough but safe* amount of resource adjustment ensures a secured application performance. To leverage the decoupling of performance prediction from the centralized resource scheduling, each node agent needs to implement fine-grained but strong isolation mechanism. In effect, hardware operations are enabled by software-defined technologies. For example, Cache Allocation Technology (CAT) [18] provides software-programmable control over the amount of cache space that can be consumed by a given thread or process.

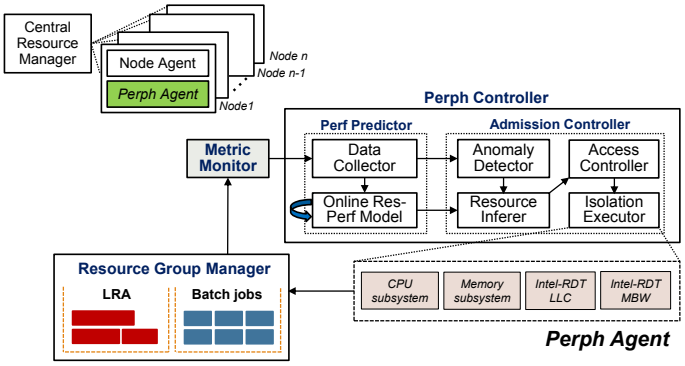

Fig. 1. Architecture overview of Perph

B. Architecture and Methodology

Decentralized Architecture. We present Perph, a runtime agent on a per node basis, that decouples ML-based performance prediction and resource inference from centralized scheduling. Fig. 1 outlines the proposed architecture. Perph is a runtime agent that runs with system agent and gauges all application performance and adaptive resource allocation at runtime. Perph is loosely couple with the central resource manager and agents on other nodes. In the following subsection, we describe objectives and designs of core Perph components.

Metric Monitor. We exploit sensitivity of applications to multi-resources to establish performance prediction. To achieve this, *Metric Monitor* aggregates application fingerprint and system-level performance metrics including CPU, memory, Last Level Cache (LLC), memory bandwidth (MBW) and number of running threads, etc. They are enabled by Intel-RDT and obtained from resource group manager. The aggregated metrics will be collected by *Data Collector* and stored in local time-series database for modeling and ad-hoc queries.

Performance Predictor. To target **[R1]**, Perph employs online machine learning mechanism to resolve model aging problem. For warm bootstrap in the early stage, we use offline and supervised learning to train the initial *Res-Perf Model*. Merely a small volume of profiling trace collected from the cluster are leveraged for the learning. We take multiple resources into account and collect not only conventional resource usage such as CPU and memory, but also other fine-grained counters including Last Level Cache (LLC) and memory bandwidth (MBW), etc. Intel-RDT technique is rapidly developed so that such metrics can be precisely monitored and collected. To solve model aging problem, model evolution is a must on the arrival of workloads. Assuming an even distribution of job arrival and job types among different nodes, Perph agent on

TABLE I
PARAMETER SETTING-UP

Resource Type	Value Range		
	UpperLimit	*LowerLimit*	*StepSize*
CPU (Available time, ms)	500,000	50,000	20,000
Memory (MB)	100	10	10
LLC (Cache way)	11	1	1
MBW (Occupancy%)	100%	10%	5%

each node can individually start and update their model. We discuss how the online model is evolved in Section III.

Runtime Admission Controller. To achieve **[R2]**, A runtime admission controller identifies the right time for resource adjustment, admits application to a specific portion of the node resources and carries out an isolated execution for a given application. More specifically, *Anomaly Detector* can timely pinpoint a performance degradation via LSTM time-series analysis and determine when and which application need to be re-allocated resources. Once abnormal performance counter or load is detected, *Resource Inferer* conducts a gradient ascend based inference to work out a proper slice of resources, towards dynamically rescuing the degraded performance. Upon receiving an updated re-allocation, *Access Controller* re-assigns a specific portion of the node resources to the affected application. Eventually, *Isolation Executor* enforces resource manipulation and ensures performance isolation across applications. Specifically, we use cgroup *cpuset* and *memory* subsystem to control usage of CPU and memory while leveraging Intel-RDT technology to underpin the manipulation of LLC and MBW. For fine-granularity management, we create different groups for LRA and batch jobs when the agent starts. The details can be found in Section IV.

III. ONLINE PERFORMANCE PREDICTION

Different LRAs may exhibit different sensitivity characteristics to the allocated resources. Database applications are essentially underpinning most back-end and front-end services [19]. As an example, we mainly focus on data-intensive workloads as our target LRA. Particularly, as MySQL own the highest market share (38.90%) in Database Management System (RDBMS) areas [20], we are motivated to use MySQL as the demonstration object. In this section, we discuss performance indicator and introduce an online prediction optimization to overcome time complexity involved in pure offline modeling.

A. Performance Indicator

Instructions Per Cycle (IPC) and Million Instructions Per Second (MIPS) are two typical performance indicators in performance engineering and performance analysis. [21][22] demonstrate that IPC is closely related to performance of latency-sensitive applications. MIPS, likewise, is another alternative since it can be approximately calculated through the production of CPU frequency and IPC.

Nevertheless, we adopt MIPS as the indicator mainly due to precision consideration. In reality, when computational workloads change, CPU frequency and the number of clock cycles consumed tend to vary due to frequency conversion or over-clocking techniques. Since IPC is highly dependent on the number of cycles, its accuracy cannot be completely guaranteed. Additionally, cycle-based metric is difficult to capture performance interference especially when an application is interrupted by I/O or network bandwidth. This is because the clock cycle will not count once it is suspended while waiting for I/O operations. By contrast, MIPS is relatively stable if external environment is unchanged, which can significantly reduce the value fluctuation and data noises.

Therefore, the main objective is to depict relationship between multi-dimensional vector $(R_{CPU}, R_{mem}, R_{LLC}, R_{MBW}, C_{app})$ and consequent performance V_{mips} where R_i represents a specific resource quota and C_{app} denotes current workload.

B. Model Selection

Firstly we use offline training to demonstrate the process of model selection. We need to determine appropriate value boundary and step for each resource dimension. In fact, resource allocation over upper boundary will no longer improve performance while severe violation is likely to manifest under the lower bound. Profiling beyond the boundaries are completely meaningless.To reduce the time cost, we temporarily limit the the amount of sample data for offline training. Detailed parameters used in the profiling are shown in Table I. In our context, independent variables used to predict MIPS are those resources that can be independently allocated. There are a variety of regression algorithms to be potentially applied into system, including Linear Regression, k-Nearest Neighbor (KNN), Adaboost, ElasticNet and Gradient Boost Regression Tree (GBRT), etc. We evaluate metrics such as Root Mean Square Error (RMSE), Mean Absolute Error(MAE), Median Absolute Error, R^2 (coefficient of determination) and other fitting effect measurement indicators when comparing different models and selecting the most suitable one. Meanwhile, to avoid over-fitting, we further perform k-fold cross-validation by dividing training data into 10 partitions, randomly selecting 9 partitions as training set and keeping the remaining one as test set in each training.

As shown in Table II, GBRT has the smallest multiple error indexes and the highest R^2, indicating its minimal prediction error. We also observe a stable prediction effectiveness in GBRT with merely 1.2 RMSE deviation. Hence, we use GBRT in the following work.

C. Online GBRT Modeling

Warm Bootstrapping and Online Calibration. Although GBRT can obtain a precise model to predict performance, it takes dozens of minutes to do so which is unacceptable in real task scheduling system. To shorten the bootstrap whilst getting a good-enough model, we warm-up the online learning with low-cost offline training by only adopting 10% original sampling points. We then use an online optimization for GBRT – online GBRT (OGBRT) starts from the offline model but continuously updates model parameters when incoming

TABLE II
MODEL SELECTION AND COMPARISON

Modeling Algorithm	Indicators			
	RMSE	***MAE***	***Med Abs Err***	R^2
Linear Regression	228.203	224.675	183.368	0.9203
KNN	811.536	539.308	429.582	0.3684
Adaboost	254.416	198.598	157.454	0.9379
ElasticNet	550.409	473.549	462.452	0.7095
GBRT	124.098	79.322	51.107	0.9852
OGBRT	114.137	63.187	35.79	0.972

workloads are executed in the system. Collected traces can boost the timely calibration based on the initial model.

Online Algorithm Design. In GBRT, weak learners measure the error calculated in each node of the regression tree, use a function $\sigma : \mathbb{R}_n \to \mathbb{R}$ with a threshold ϕ to split the node and eventually return values λ^l and λ^r. We can get the optional split represented by triple $(\phi, \lambda^l, \lambda^r)$ after minimizing the error in Eq. 1,

$$\theta(\phi) = \sum_{i:\sigma(\mathbf{x}_i)<\phi} w_i^j (y_i^j - \lambda^l)^2 + \sum_{i:\sigma(\mathbf{x}_i)\geq\phi} w_i^j (y_i^j - \lambda^r)^2 \quad (1)$$

where w_i^j and y_i^j respectively represent the weight and response of x_i in the k-th iteration. In formal, loss function and w_i^j is given in Eq. 2 [23] where w_i^j can be seen as a measure of the "influence" of the k-th estimation.

$$\begin{aligned} L &= \sum_{i=1}^{N} log[1 + exp(-2y_i f(\mathbf{x}_i)))] \\ w_i^j &= exp(-2y_i f_{k-1}(\mathbf{x}_i)) \end{aligned} \quad (2)$$

We can use a generic framework to underpin the learning in OGBRT. Given samples can be provided sequentially by *metric monitor* in Perph agent, we update each weak learner in an online manner, during which samples are used for training until previous data is no longer available. Suppose there is a regressor at the current time t, its error at a GBRT node m in a weak regressor is defined in Eq. 3

$$\theta(\phi_{t,m}) = \sum_{i=1}^{N_m} w_{t,i}(y_i - \lambda_{t,m})^2 \quad (3)$$

$$\lambda_{t,m} = \begin{cases} \lambda_{t,m}^l, \text{ if } \sigma(\mathbf{x}_i) < \phi \\ \lambda_{t,m}^r, \text{ otherwise} \end{cases} \quad (4)$$

where N_m is the number of examples that fall on node m. The objective of our OGBRT is to learn new parameters $\phi_{t+1,m}$ and $\lambda_{t+1,m}$ given n new samples are generated from *monitor* to the regressor through optimization of the following function (Eq. 5):

$$\begin{aligned} \theta_{t+1}(\phi_{t+1,m}) &= \sum_{i=1}^{N_m+n} w_{t+1,i}(y_i - \lambda_{t+1,m})^2 \\ &= \sum_{i=1}^{N_m+n} w_{t+1,i}(y_i - (\lambda_{t,m} + \Delta\lambda))^2 \end{aligned} \quad (5)$$

Once $\phi_{t+1,m}$ is known at t, $\lambda_{t+1,m}$ can be firstly optimized. However, $(w_{t+1}, i, y_i)_{i=1\ldots N_m}$ is unavailable before regressor f_{t+1} is established, which makes it impossible to minimize the θ_{t+1}. Therefore, we further improve the performance of regressor based on new n samples based on the current regressor. We get Eq. 6:

$$\begin{aligned} \theta_{t+1}(\phi_{t+1,m}) = \sum_{i=1}^{N_m+n} \{ & w_{t+1,i}((\Delta\lambda)^2 - 2(y_i - \lambda_{t,m})\Delta\lambda) \\ & + w_{t+1,i}(y_i - \lambda_{t,m})^2 \} \end{aligned} \quad (6)$$

According to *quadratic formula*, we can easily get Eq. 7:

$$\Delta\lambda_{t,m} = \begin{cases} \Delta\lambda^l(\phi_{t+1,m}), \text{ if } \sigma(\mathbf{x}_i) < \phi \\ \Delta\lambda^r(\phi_{t+1,m}), \text{ otherwise} \end{cases} \quad (7)$$

where

$$\Delta\lambda^l(\phi_{t+1,m}) = \frac{\sum_{i:\sigma(\mathbf{x}_i)<\phi_{t+1,m}}^{N_p+n} w_{t+1,i} y_i}{\sum_{i:\sigma(\mathbf{x}_i)<\phi_{t+1,m}}^{N_p+n} w_{t+1,i}} - \lambda_{t,m}^l \quad (8)$$

$$\Delta\lambda^r(\phi_{t+1,m}) = \frac{\sum_{i:\sigma(\mathbf{x}_i)\geq\phi_{t+1,m}}^{N_p+n} w_{t+1,i} y_i}{\sum_{i:\sigma(\mathbf{x}_i)\geq\phi_{t+1,m}}^{N_p+n} w_{t+1,i}} - \lambda_{t,m}^r \quad (9)$$

Because $(w_{t+1,i}, y_i)_{i=1\ldots N_m}$ is unknown, we recursively get λ by Eq. 10 and Eq. 11, where $0 < \alpha < 1$ represents the learning rate detailed in [24].

$$\begin{aligned} \lambda^l(\phi_{t+1,m}) &= \frac{\sum_{i:\sigma(\mathbf{x}_i)\geq\phi_{t+1,m}}^{N_p+n} w_{t+1,i} y_i}{\sum_{i:\sigma(\mathbf{x}_i)\geq\phi_{t+1,m}}^{N_p+n} w_{t+1,i}} \\ &\approx (1-\alpha)\lambda_{t,m}^l + \alpha \sum_{i=N_{m+1}}^{N_{m+n}} w_{t+1,i} y_i \end{aligned} \quad (10)$$

$$\begin{aligned} \lambda^r(\phi_{t+1,m}) &= \frac{\sum_{i:\sigma(\mathbf{x}_i)\geq\phi_{t+1,m}}^{N_p+n} w_{t+1,i} y_i}{\sum_{i:\sigma(\mathbf{x}_i)\geq\phi_{t+1,m}}^{N_p+n} w_{t+1,i}} \\ &\approx (1-\alpha)\lambda_{t,m}^r + \alpha \sum_{i=N_{m+1}}^{N_{m+n}} w_{t+1,i} y_i \end{aligned} \quad (11)$$

Finally, to put things together, Alg. 1 describes the holistic OGBRT when n new examples are given for the model update. Initial experiment shows the precision improvement in OGBRT (see Table II) – the overall effectiveness of OGBRT can nearly reach the same level of offline learning based on big sampling volume.

Algorithm 1 Online GBRT Algorithm

Input: regressor in time t: $f_0 = f_t, \{\mathbf{x}_i, y_i\}_{i=1:n}$
Output: f_{t+1} – updated regressor at time t+1

```
Initialize θ_m ← ∞ , φ_{t+1,d} ← φ_{t,d}
for k = 1...M do
    w_i = exp(−2 y_i f_{k−1}(x_i)), i = 1...n
    Pick examples falling on each node in the gbrt tree.
    for each node d from root do
        for each φ_{t+1,d} do
            λ^l_{t+1,d}, λ^r_{t+1,d}, θ_{t+1,d} ← Using Eq.10, Eq.11 and Eq.6
            if θ_{t+1,d} < θ_m do
                θ_m = θ_{t+1,d}; φ*_{t+1,d} = φ_{t+1,d}
        end for
        Get (λ^{l*}_{t+1,d}, λ^{r*}_{t+1,d}) using φ*_{t+1,d}
    end for
end for
```

IV. Runtime Admission Control

Herein we present how resource inference and runtime admission controller work in Perph. As aforementioned, once Perph detects performance anomalies among running applications via LSTM time series analysis, dynamic resource adjustment will be enforced after resource re-evaluating. Inferring a *just enough* amount is essential to safely rescue LRA's performance whilst keeping the system compacted. Additionally, each Perph agent needs to implement fine-grained but strong isolation mechanism. Due to the limited space available, we omit the detailed discussion of anomaly detection in this paper, but will include them in the coming preprint.

A. Resource Inference Based on Gradient Ascending

Once performance interference manifests, we essentially need performance remediation through adjusting resource allocation. GBRT model is the cornerstone that we can leverage to find the optimal allocation scheme. However, the option is not unique – there are probably multiple options that can achieve the same effectiveness due to the diverse sensitivity of application performance to different resource dimensions. To improve node utilization, we necessarily find a scheme with minimal resource change but maximal performance benefit. The benefit in this context indicates that performance can be rescued to the desired level. Specifically, we calculate the partial derivative at each resource dimension and form the gradient. Actually, gradient descent algorithm is typically used to find the quickest path to reach the minimized loss function. Inspired by this, likewise, we employ a gradient ascent algorithm to continually boost the performance – At each decision point, we compute the gradient of f along different resource dimensions (Eq. 12) and reallocate resource with the largest gradient, thereby ensuring the quickest performance remediation.

$$\nabla f(r_k^1, \ldots, r_k^m) = (\frac{\partial f}{\partial r_k^1}, \ldots \frac{\partial f}{\partial r_k^m})\mathbf{r} \tag{12}$$

$\mathbf{r}$ is the base vectors in multi-dimensional Cartesian coordinates consisting of unit vectors $\mathbf{i}$ of different direction. Numerically, gradient can be calculated by aggregating partial derivatives and res-perf model provisions the estimated value that can be directly substituted into Eq. 13.

$$\nabla f(r_k^i) = \frac{\partial f}{\partial r_k^i} = \frac{f(r_k^i + \Delta r^i) - f(r_k^i)}{\Delta r^i} \tag{13}$$

If the learning rate α remains unchanged during the gradient ascending, over-allocation may be obtained by skipping the optimal value that satisfies the condition. To this end, we gradually decrease α based on simulated annealing, where t_0 and t_1 are constants to control change rate of α. Then we adjust the allocation by factor α indicating the adjustment step (Eq. 14).

$$r_{k+1}^i = r_k^i + \frac{t_0}{t_1 + k} \nabla f(r_k^i) \tag{14}$$

Algorithm 2 Resource Inference Algorithm

Input: $tasks$ – current number of threads in the target application;
$MIPS$ – expected performance of the target application;
ε – threshold for iteration termination;
$\mathbf{R} \in \mathbb{R}_4$ – the resource vector needed to be adjusted;
$\boldsymbol{\theta_r}$ – Direction vector for resource adjustment;
β – control the step length in the gradient direction
Output: R – Optimal resource quota for Input MIPS

1: **while** $(|V_R - MIPS| > \varepsilon)$ **do**
2: **for** $i \in (CPU, MEM, LLC, MBW)$
3: $\frac{\partial V}{\partial r^i} \leftarrow \frac{V(r^i + \Delta r^i) - V(r^i)}{\Delta r^i}$
4: **end for**
5: $\boldsymbol{\theta_r} \leftarrow \boldsymbol{\theta_r} + \frac{t_0}{t_1 + iter} \frac{\partial V}{\partial \mathbf{r}}$; $\quad \mathbf{R} \leftarrow \mathbf{R} + (\boldsymbol{\theta_r} * \beta)\mathbf{R}$
6: $iter \leftarrow iter + 1$
7: **end while**
8: **return R**

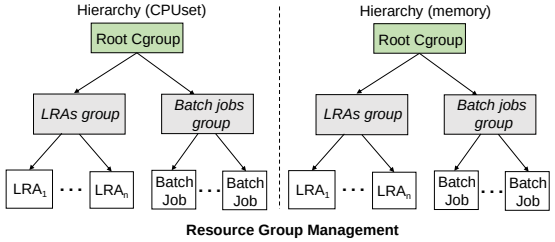

Fig. 2. Cgroup-based resource isolation

The iterations will be terminated until estimated performance surpasses the desired one. Subsequently, all resource series $[\mathbf{r_1}, \ldots, \mathbf{r_k}, \ldots, \mathbf{r_K}]$ are accumulated before being passed to the access controller. The procedure are described in Alg. 2.

B. Adaptive Isolation and Execution

The primary functionality is to form a complete set of operational instructions for resource reallocation in terms of CPU, memory, LLC and MBW. Since allocation of LLC and MBW have to rely on Class of Service (CLOS), their allocation should be conducted after CPU and memory. This ensures resource groups with different CPU affinity have different portions of LLC and MBW.

CPU and Memory. Perph implements CPU isolation by using cgroup cpuset subsystem instead of cgroup cpu subsystem (used by [25][26][27][28]). In reality, time slice control in cpu subsystem may cause frequent switches between CPU cores. CPU sharing would cause CPU cache contention in hyper-threading, giving rise to non-negligible system overheads. Also, cpu subsystem schedules CPU access to each cgroup using either Completely Fair Scheduler (CFS) (e.g., by the default on Linux and Docker) or Real-Time Scheduler (RT). Batch jobs, however, may preempt CPU resources owned by LRAs in this scenario, potentially resulting in SLO violation. In Perph, we set CPU affinity for different process group, thereby guaranteeing performance at all time. Another consideration when performing CPU logical core binding is that we attempt to allocate logical cores of the same CPU slot to a given long-running application. We also round the predicted CPU usage value up when setting the value of $CPUset.cpus$ to directly meet minimum performance require-

ment. Moreover, we can further partition resource by using CPU subsystems based on the CPU logical core binding. Perph leverages memory cgroup subsystem to limit the amount of available memory to the LRA by setting *memory.limit*. To completely utilize node resources, we allow batch job group for having the remaining resources. Fig. 2 outlines how Perph agent manages CPU and memory with the group hierarchy.

LLC and MBW. Intel RDT supports a mechanism to monitor and control access to LLC and MBW to avoid resource starvation and consequent performance degradation. We mainly rely on Cache Allocation Technology (CAT) [18] and Memory Bandwidth Allocation (MBA) [29] to control these admissions. Specifically, different cache areas will be distinguished by CLOS and members in each CLOS can only access to a specific portion of cache within its pertaining area. Furthermore, RDT splits LLC into equal-sized logical partitions and Cache Bit Mask (CBM) is used to indicate the access to each partition. We adopt this mechanism to select particular LLC ways for different CLOSs. Perph will dynamically generate bit masks according to the result of runtime resource re-allocation. Likewise, MBA provides how memory bandwidth is distributed across running applications. Based on the CPU affinity settings in different resource groups, Perph binds different resource groups to different CLOS and set corresponding MBW to them.

V. System Implementation

We integrated proposed Perph mechanisms with Node Manager (NM) in Apache YARN. We also modified relevant modules in Application Master (AM) and message protocols between AM, NM and the central Resource Manager (RM).

Firstly, we customize different AMs to underpin the lifecycle of LRA and batch jobs. To differentiate the container type, a tag will be labeled by AM and piggybacked when requesting resources to RM and sending execution plans to NM. RM is aware of container type in the procedure of registration and further resource allocation. Meanwhile, LRA AM is responsible for DAG management encompassing both topology configuration and dependencies among different components in LRA. Because we encapsulate all LRA containers in Docker container, AM also needs to deal with image storage and meta information maintenance. Hence, once AM gets specific resources granted by RM, it will leverage the self-contained DAG information and repository of each container to launch relevant containers on corresponding NM.

We also modified RM to best serve Perph mechanisms. We grant different priorities to LRA and batch jobs. Particularly, AM of LRA is prioritized and given privilege to directly oversubscribe idle resources or preempt resources from low priority batch jobs when current available resources in the node are not enough. The procedure allows for quick resource adjustment used in Perph and makes it free from RM temporarily. This is largely backed by our previous work [30]. To synchronize resource usage, NM will coordinate the oversubscribed and preempted resource slices with RM for timely updating.

TABLE III
Experiment Environment

OS	Kernel version	Linux v4.9.0-6-amd64
	Release version	Debian 4.9.82
Hardware	CPU version	Intel-Xeon(R)-Silver 4110CPU@2.10GHz
	Physical CPU cores	16 (8/sockets * 2sockets)
	Logical CPU cores	32 (16/sockets * 2sockets)
	LLC	11MB
	memory	187GB

The most important integration is to align Perph Agent with current NM for performance prediction, resource inference and isolated execution. We implement a resource group manager (RGM), metric collector (MCo) and Perph controller (PCo) in native NM. Upon arrival of new tasks, RGM ensures smooth establishment of subgroups and the initialization (e.g., mounting) of cgroup subsystems in the bootstrapping phase. Meanwhile, RGM is also responsible for hierarchical management. MCo traces and collects required statuses and metrics at container, application and node level using perf counter tools, and then feeds the trace data to PCo. PCo is the core integration in NM that encompasses data receiving, local model training, model synchronization and evolution, and runtime management we discussed in previous sections. Particularly, PCo is an individual implementation on per node basis that runs independently of RM. Detection module is developed along with PCo and runs in background to capture performance and trigger runtime resource reallocation.

VI. Evaluation

A. Experiment Setup

Environment. To illustrate the general applicability, we deployed Perph into Apache YARN to demonstrate improvements in performance guarantee when different types of workloads are co-located. Evaluation was performed in a 32-node cluster. Each node is Intel-rdt enabled and equipped with 187GB DRAM, 11MB LLC and 16 CPU physical cores running at 2.10GHz. More detailed can be found in Table III.
Workloads. To emulate realistic applications in cloud datacenters, we use a mixture of workloads, encompassing data-centric applications such as latency-sensitive database and data-streaming applications, and background batch jobs. In our experiment, we select MySQL and Kafka as representatives. Meanwhile, to generate consecutive and CPU-heavy background noises, we use map-reduce batch jobs which are insensitive to latency but merely account for end-to-end completion time. We submit PI jobs and specific parameters such as the mapper number and sampling time to make them run over the whole running period of long-running applications. We generate 50 millions message workloads (each of which is 1KB) to Kafka within the same environment. Additionally, database workloads can be generally categorized into Online Transaction Processing (OLTP) and Online Analytic Processing (OLAP). We considered both of them in this study:

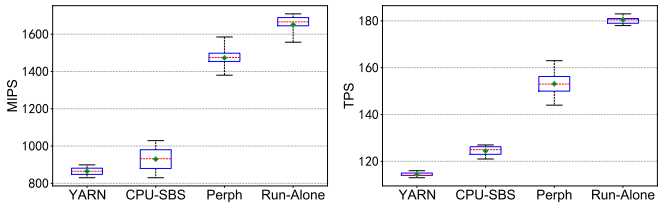

Fig. 3. TPC-C MIPS and TPS

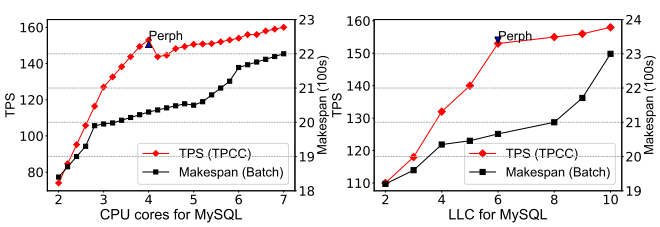

Fig. 4. Effectiveness validation of resource allocation in TPC-C benchmarking

- **OLTP:** OLTP workloads are characterized by a large number of interactive database operations with high concurrency requirements. We use i) TPC-C that queries from retail database and involves a mixture of five concurrent transactions with different types and complexity either executed on-line or queued for deferred execution. ii) MySQL's official lightweight testing tool mysqlslap that emulates a large number of client connections hitting the database server simultaneously.
- **OLAP:** OLAP targets those decision support procedure in enterprises. Due to the large amount of data queries, it has high requirements for IO performance. TPC-H, a data-warehousing benchmark, generates business analytic queries to a database of sales data. We adopted different queries using a 5GB database in the evaluation.

Baseline and Methodology. We submit long-running MySQL and Kafka application into YARN and consecutively submit above workloads. To reduce result deviations, we repeat the same experiment for 10 times. We compare Perph against the following execution schemes:

- **Native YARN**: MySQL and batch jobs are co-located under isolation mechanism provided by native YARN Node Manager.
- **CPU-SBS**: MySQL and batch jobs are co-located under isolation mechanism provided by cpu subsystem without LLC and MBW control and isolation.
- **Run-Alone**: We collect metrics when MySQL is running alone, where theoretically no interference is generated. It is used as the baseline to evaluate the degree of performance interference caused by co-location.

Metrics. We use the following metrics in our experiments:

- **MIPS**: we measure the Million Instructions per Second of both database and Kafka streaming;
- **Transactions per Second (TPS):** It indicates the throughput of database transactions;
- **Query Per Second (QPS) or Query Per Hour (QPH)**: it indicates the database query efficiency;

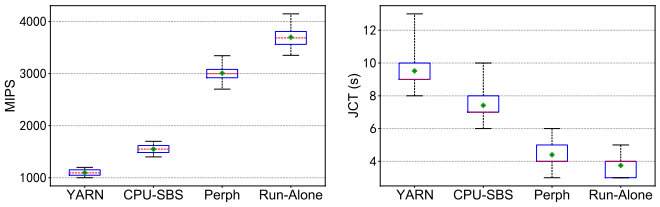

Fig. 5. mysqlslap MIPS and job completion time

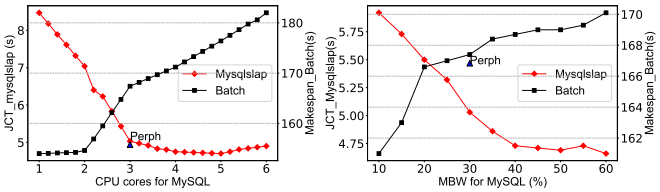

Fig. 6. Effectiveness validation of resource allocation in mysqlslap

- **Throughput and Latency**: we calculate the number of messages per second of and latency of Kafka.
- **Makespan of batch jobs:** we measure end-to-end execution time (e2e time) of all submitted batch jobs.

B. Database OLTP and Batch Job Co-location

In this experiment, we employ TPC-C and mysqlslap to conduct a stress test and monitor the performance deviations.

TPC-C Benchmarking. We generate a database that encompasses 10 warehouses, each of which corresponds to 10 districts and each district contains 3k consumers. Afterwards, we set the concurrency to be 100 and run for 1,800 seconds. To ensure MapReduce PI job can run during the whole duration, the job is set to have 500 mappers and each mapper contains 5 millions sampling points. We count the completed transaction number of 5 different TPC-C businesses in repeated 10 times experiments.

MIPS and TPS Comparison. Fig. 3 illustrates that Perph far outperforms YARN and CPU-SBS. Specifically, MIPS of Perph is 1.70x and 1.58x times that of YARN and CPU-SBS, while TPS is improved by 35% and 23% compared with YARN and CPU-SBS. By contrast, CPU-SBS can only reach no more than 9% improvement of TPS compared against YARN. The main reason for the discrepancy is due to the increased overhead of process scheduling among CPUs by using cpu subsystem. As there is no grouping mechanism for CPU logical cores, fine-grained resource partitioning for LLC and mbw of the CPU cannot be performed in CPU-SBS.

Effectiveness Analysis. Firstly, the completion time of performance isolation using Perph is less than 1 second due to the pre-established performance prediction model. Isolation methods based on real-time performance feedback take a long time to complete isolation. For example, Heracles [10] requires a delay analysis using more than 15 seconds each time, and a 5-minute performance observation is reserved before the performance isolation is enforced. Furthermore, we demonstrate Perph can produce the optimal resource quota for performance isolation whilst sparing sufficient resources for other batch jobs. Herein, we observed Perph made a

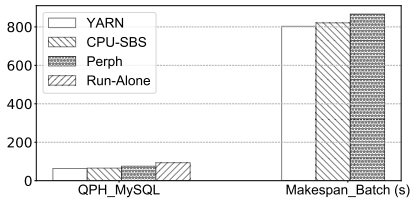

Fig. 7. QPH of TPC-H and makespan of batch jobs

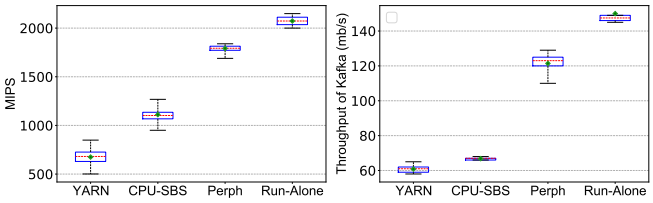

Fig. 8. MIPS and throughput of Kafka using different isolation schemes

runtime resource allocation $R(CPU, memory, LLC, mbw)$ by (4Cores, 300MB, 6, 45%) at a certain time. We slightly adjust the allocated CPU cores from 2 to 7 and LLC from 2 to 10 while keeping other dimensions unchanged. As shown in Fig. 4, overall TPS does not always increase with the increment of CPU cores allocated to MySQL. In reality, when the number of CPU cores is increased to 6, TPS even decreases by 7% due to the overhead growth in process switching between CPU cores. This demonstrates Perph can effectively find the option results. Meanwhile, the increase of CPU allocation to the database will constantly increase completion time of batch jobs because fewer CPUs cores can be over-subscribed to those jobs. A similar phenomenon manifests when we varies the LLC allocation. These indicate that Perph prioritizes the performance of LRA, manages to pick the optimal allocation scheme which make the node resource best used.

Mysqlslap Benchmarking. Mysqlslap creates short-lived test cases with adjustable concurrencies. In this experiments, we submit 200k SQL queries with 25 concurrencies (i.e., *mysqlslap -a -c 25 –number-of-queries 200000 -i 5*). Meanwhile, the batch job starts with 20,000 mapper, each of which has 100,000 sampling points to cover the whole duration.

MIPS and JCT Comparison. As shown in Fig. 5, Perph can substantially improve MIPS and transactional throughput against native YARN and CPU-SBS. Specifically, average MIPS of Perph is 2.74x and 1.94x times that of YARN and CPU-SBS. Correspondingly, the holistic time to complete all transactions on average can be reduced by 50.6% and 37.9% respectively against YARN and CPU-SBS. Even compared against Run-Alone scenario, MIPS is just slightly declined by 14.2% and the completion time merely increases by 16.5%. This is because Perph exploits online performance prediction model for provisioning timely adjustment to best target performance requirement. Cgroup cpuset subsystem used in Perph can also provide fine-grained resource isolation but reduce overheads of switching between CPUs.

Effectiveness Analysis. we record the real-time resource allocation $R(CPU, memory, LLC, mbw)$ at a certain time point: (3Cores, 350MB, 4, 30%). In a similar way to TPC-C benchmarking, we slightly change the possible value of resource allocation and examine the resultant performance variations. Particularly, the number of CPU cores varies from 1 to 6 while memory bandwidth varies from 10% to 60%. It is obvious from Fig. 6 that the allocated scheme given by Perph is the optional choice taking into account the impact on overall execution time. Specifically when the number of CPU cores increases, the execution time is reduced but the improvement is merely 4% that can be relatively negligible. This indicates that the resource allocation vector generated by Perph is accurate and *good enough* to target the performance goal at runtime. Similar phenomenon can be found when the memory bandwidth is changed. In the meantime, we can also find a growth in the makespan of batch jobs with the increment of MySQL's resource allocation. Due to the descending available resources to batch jobs, the overall makespan is naturally extended. In effect, the gradient ascending resource inference will achieve the fastest approaching to the target performance and once the effect of performance rescue slows down, Perph prefers to spare available resources to batch jobs rather than allocating more resources for further performance gain.

C. Database OLAP and Batch Job Co-location

To demonstrate the wide applicability, we use TPC-H, generate a 5G data warehouse (equivalently scale factor 5) and run 22 queries respectively to conduct multiple concurrent performance tests. We make the background batch job 2k mapper, each of which contains 100k sampling points.

As shown in Fig. 7, Perph can improve QPH by 19.0% and 15% compared with YARN and CPU-SBS respectively. This is also due to the timely and just-enough resource inference with fine-grained isolation. QPH in Perph, however, decrease by roughly 19% against that of database runs alone. Since less resources can be spared to batch jobs, their makespan is slightly extended in Perph. In fact, TPC-H benchmark has heavy requirements of disk I/O, but the limited IO isolation in Perph currently restricts the capability of dealing with IO interference and thus improvement of QPH. We plan to enhance Perph via investigating in IO isolation in our future work.

D. Data-streaming LRA and Batch Job Co-location

As demonstrated in Fig. 8, a similarity between MIPS and throughput manifests. The phenomenon is also similar to results we obtained in OLTP benchmarkings. Although there are 13.6% MIPS and 14% throughput degradation compared with Run-Alone scenario, performance of Perph is substantially improved compared with other isolation schemes – average MIPS is 2.65x times and 1.61x times that of YARN and CPU-SBS. Likewise, througput on aveage is 2.01x and 1.82x times that of YARN and CPU-SBS. The results signify our proposed mechanism have positive impacts on the performance guarantee of data-streaming applications.

Fig. 9 depicts the latency of Kafka workload. To be precise, we measure median, 90th, 95th, 99th percentile response time

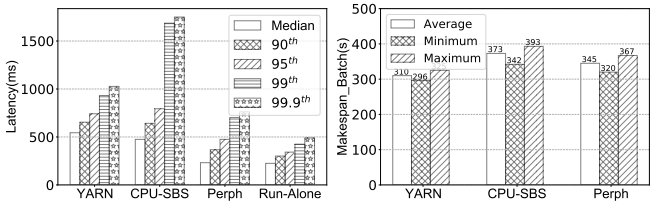

Fig. 9. Latency of Kafka and makespan of batch jobs

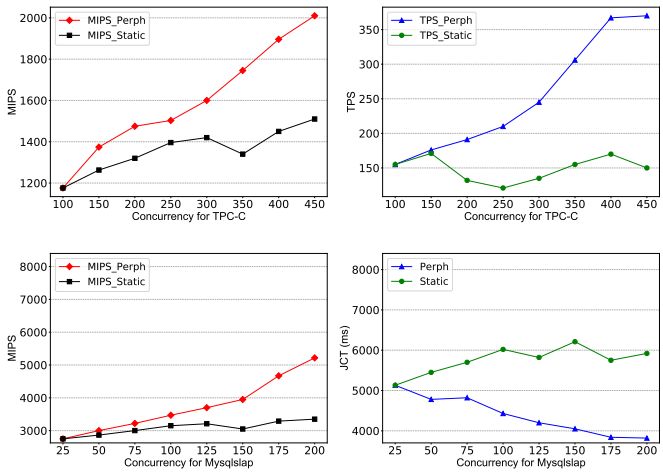

Fig. 10. Scalability evaluation

among all requests. It is observable that Perph can achieve 57.4% and 51.2% reduction regarding the median latency compared with YARN and CPU-SBS. It is worth noting that the long tailed latency can be significantly mitigated by Perph – 99th percentile latency can be reduced by 24.7% and 58.4% against YARN and CPU-SBS. Interestingly, even though CPU-SBS can slightly drop the median latency, long tail latency still manifests – a 70.4% growth can be found compared against YARN. This is largely due to the high overhead derived from switches between CPU logical cores. Moreover, we compare the makespan of co-residential batch jobs to evaluate how different isolation scheme influences on them. Results demonstrate that CPU-SBS delays the makespan by 20.3% while Perph will only increase batch execution by 11.3%.

E. Scalability Evaluation

We evaluate how Perph is scalable enough with increased concurrency level. In this experiment, we run the same TPC-C and mysqlslap (used in Section VI-B) and vary the number of transaction concurrencies from 100 to 450 for TPC-C and from 25 to 200 for mysqlslap. We compare Perph with YARN which uses static resource allocation without runtime resource adjustment. As shown in Fig. 10, MIPS of Perph will grow linearly with the increment of concurrencies. By contrast, the MIPS of TPC-C in YARN is almost stable, even with slight decrease. This is because resource inference based dynamic reallocation will be conducted when increased system loads are detected. More resources will be properly allocated to the application, thereby maintaining the desired performance level. Similarly, Perph can also guarantee the throughput at runtime compared with the degraded throughput in YARN. Likewise, MIPS for mysqlslap workloads has the same phenomenon. Due to sufficient resource re-allocation, the overall mysqlslap makespan can be even shorten by Perph.

F. System overhead

In this experiment, we analyze the system overhead of Perph. First, we observe the resource overhead of Perph compares to Apache YARN. For the same workloads, compared with NM in Apache YARN, the average memory usage of Perph agent increased from 3083MB to 3165MB , an increase of 2.65%, while the average CPU usage of Perph increased by 5% due to online performance prediction and resource inference. Second, we calculate the time cost of using Perph for performance isolation, which is primarily due to resource inference and the execution of resource isolation. The results of our multiple observations show that the average time cost is less than 300ms, which is acceptable since Perph performs resource isolation only when performance degradation is detected.

VII. Related Work

Resource management. Resource management systems in shared clusters can be divided into two categories: centralized and decentralized systems. Centralized approaches assign resources base on user requests [28][31][4] or framework offers [32]. Multiple resources will be negotiated among diverse applications through a central resource manager. To make the procedure fair and avoid resource starvation, DRF [33], capacity scheduling [34] or fairness scheduling [35] are adopted in the resource sharing among multiple jobs. Decentralized approaches [10] [36] [30] are developed for clusters that expect a high throughput or high cluster utilization. They mainly optimize task placement to enable that very short, sub-second tasks can quickly access to idle resources. However, these systems merely best serve batch workloads and thus lack an effective mechanism for resource throttling or coordinated feedback, which leads to unawareness of performance impact on co-located LRAs. In this work, to timely guarantee LRA's performance and reduce the burden of centralized learning, we design and implement Perph within distributed agents.

ML and performance prediction in resource scheduling. Many approaches have applied machine learning (ML) to improve datacenter management and resource scheduling. Most of them focus on workload characterizing and behavior prediction. For instance, [37][38] leverage various ML methods such as SVR, random forest and extreme gradient boosting tree to predict workloads or system load changes. [39][40] employ neural network to estimate job makespan and load fluctuation. However, the limitation in those methods is their heavy dependence on offline historical information. This leads to the fact that they can hardly take runtime information into consideration and provide sufficient insights into timely calibration of workload performance. [41][11] use complicated multi-variable statistical classifiers to predict the expected interference among applications. They perform beforehand small-scale interference tests with varied levels of background

applications. However, without online model calibration, the misprediction may lead to high system overheads. Also, the pair-based profiling results in huge experimentation cost. [13] uses performance index to depict contention at the time of resource allocation. However, the feedback mechanism lacks an understanding of the relationship between multiple resources and the resulting performance – which makes it difficult to perform fine-grained controls over different resources. [12] is mainly based on offline profiling and thus time-consuming during data collection and model training. In comparison, Perph employs offline supervised learning to quickly warm up but uses online optimization for model evolution.

VIII. Conclusion and Future Work

In this paper, we present Perph, a ML-based agent on a per node basis for workload co-location. Perph is mainly composed of online performance prediction and timely resource inference mechanism and thus can overcome the time complexity and imprecision of pure offline profiling based approaches. By exploiting the sensitivity of long-running applications to multi-resources, we can approximate the relationship between resource allocation and consequential performance. We use online GBRT to enable the continuous model evolution. Once performance degradation or load spike is detected, Perph will infer a proper slice of resources, thereby calibrating safe but enough resources to the suffered application. In the future, we intend to federate individual agents and coordinate the model learning. We also plan to integrate Perph mechanism with our previous work on resource over-subscription [30] to supervise the workload co-location considering both LRA's runtime performance and batch job's throughput.

Acknowledgment

This work is supported by National Key Research and Development Program of China (2016YFB1000503), NSFC(61421003) and UK EPSRC (EP/T01461X/1). This work is also supported by Beijing Advanced Innovation Center for Big Data and Brain Computing (BDBC).

References

[1] L. A. Barroso and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis lectures on computer architecture*, 2009.
[2] J. Dean and L. A. Barroso, "The tail at scale," *Comm. of ACM*, 2013.
[3] H. Zhu and M. Erez, "Dirigent: Enforcing qos for latency-critical tasks on shared multicore systems," *ACM ISCA*, 2016.
[4] A. Verma, L. Pedrosa *et al.*, "Large-scale cluster management at google with borg," in *ACM Eurosys*, 2015.
[5] Q. Liu and Z. Yu, "The elasticity and plasticity in semi-containerized co-locating cloud workload: A view from alibaba trace," in *ACM SoCC*, 2018.
[6] S. Das, V. R. Narasayya, F. Li, and M. Syamala, "Cpu sharing techniques for performance isolation in multi-tenant relational database-as-a-service," *VLDB*, 2013.
[7] P. Delgado, D. Didona *et al.*, "Job-aware scheduling in eagle: Divide and stick to your probes," in *ACM SoCC*, 2016.
[8] P. Delgado, F. Dinu, A.-M. Kermarrec, and W. Zwaenepoel, "Hawk: Hybrid datacenter scheduling," in *USENIX ATC*, 2015.
[9] J. Leverich and C. Kozyrakis, "Reconciling high server utilization and sub-millisecond quality-of-service," in *ACM Eurosys*, 2014.
[10] D. Lo, L. Cheng, R. Govindaraju *et al.*, "Heracles: Improving resource efficiency at scale," in *ACM ISCA*, 2015.
[11] C. Delimitrou and C. Kozyrakis, "Quasar: resource-efficient and qos-aware cluster management," in *ACM ISCA*, 2014.
[12] Y. Sfakianakis, C. Kozanitis, C. Kozyrakis, and A. Bilas, "Quman: Profile-based improvement of cluster utilization," *ACM TACO*, 2018.
[13] P. Lama, S. Wang *et al.*, "Performance isolation of data-intensive scale-out applications in a multi-tenant cloud," in *IEEE IPDPS*, 2018.
[14] R. Sen and K. Ramachandra, "Characterizing resource sensitivity of database workloads," in *IEEE HPCA*, 2018.
[15] Intel resource director technology. "https://www-ssl.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html".
[16] D. Sanchez and C. Kozyrakis, "Scalable and efficient fine-grained cache partitioning with vantage," *IEEE Micro*.
[17] X. Wang, S. Chen, J. Setter, and J. F. Martínez, "Swap: Effective fine-grain management of shared last-level caches with minimum hardware support," in *IEEE HPCA*, 2017.
[18] Cmt and cat. [Online]. Available: https://www.intel.com/content/www/us/en/communications/cache-monitoring-cache-allocation-technologies.html
[19] M. Hui, D. Jiang, G. Li, and Y. Zhou, "Supporting database applications as a service," in *IEEE ICDE*, 2009.
[20] 2019 db trends. [Online]. Available: https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/
[21] X. Zhang, E. Tune *et al.*, "Cpi 2: Cpu performance isolation for shared compute clusters," in *ACM Eurosys*, 2013.
[22] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers," in *ACM ISCA*, 2013.
[23] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, 2001.
[24] B. Boris, Y. Ming-Hsuan, and B. Serge, "Robust object tracking with online multiple instance learning," *IEEE TPAMI*, 2011.
[25] Cgroups[eb/ol]. [Online]. Available: https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt
[26] Docker[eb/ol]. [Online]. Available: https://www.docker.com/
[27] Using cgroups with yarn[eb/ol]. [Online]. Available: http://hadoop.apache.org/docs/r3.0.0/hadoop-yarn/hadoop-yarn-site/NodeManagerCgroups.html
[28] V. K. Vavilapalli, A. C. Murthy, C. Douglas *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *ACM SoCC*, 2013.
[29] Resource allocation in intel® resource director technology. "https://01.org/zh/node/6485?langredirect=1".
[30] X. Sun, C. Hu, R. Yang *et al.*, "Rose: Cluster resource scheduling via speculative over-subscription," in *IEEE ICDCS*, 2018.
[31] Z. Zhang, C. Li, Y. Tao, R. Yang, H. Tang, and J. Xu, "Fuxi: a fault-tolerant resource management and job scheduling system at internet scale," *VLDB*, 2014.
[32] B. Hindman, A. Konwinski, M. Zaharia *et al.*, "Mesos: A platform for fine-grained resource sharing in the data center." in *USENIX NSDI*, 2011.
[33] A. Ghodsi, M. Zaharia *et al.*, "Dominant resource fairness: Fair allocation of multiple resource types," in *USENIX NSDI*, 2011.
[34] Capacity scheduler. [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html
[35] Fair scheduler. [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html
[36] K. Karanasos, S. Rao *et al.*, "Mercury: Hybrid centralized and distributed scheduling in large shared clusters," in *USENIX ATC*, 2015.
[37] C. Liu, Y. Shang *et al.*, "Optimizing workload category for adaptive workload prediction in service clouds," in *ICSOC*. Springer, 2015.
[38] E. Cortez, A. Bonde, A. Muzio *et al.*, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *ACM SOSP*, 2017.
[39] M. R. Wyatt II, S. Herbein *et al.*, "Prionn: Predicting runtime and io using neural networks," in *ACM ICPP*, 2018.
[40] Q. Yang, C. Peng, and Others, "A new method based on psr and ea-gmdh for host load prediction in cloud computing system," *TJSC*, 2014.
[41] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *ACM SIGPLAN Notices*, 2013.