Poster: Flex: Closing the Gaps between Usage and Allocation

Tan N. Le
SUNY Korea, Stony Brook University
tan.le@stonybrook.edu

Zhenhua Liu
Stony Brook University
zhenua.liu@stonybrook.edu

ABSTRACT
Most of data centers are significantly underutilized. One of the major reasons is the big gaps between the real usage and the provisioned resources. In this paper, we first conduct an in-depth analysis of a Google cluster trace to unveil the root causes for low utilization and highlight the great potential to improve it. We then developed an online resource manager Flex to maximize the cluster utilization while satisfying the Quality of Service (QoS). Large-scale evaluations show that Flex admits up to 1.74× more requests and 1.6× higher utilization compared to tradition schedulers while maintaining the QoS. The full version of this paper is [4].

CCS CONCEPTS
• Applied computing → Data centers.

KEYWORDS
Resource Allocation, Task Schedulers, Distributed Systems.

ACM Reference Format:
Tan N. Le and Zhenhua Liu. 2020. Poster: Flex: Closing the Gaps between Usage and Allocation. In The Eleventh ACM International Conference on Future Energy Systems (e-Energy’20), June 22–26, 2020, Virtual Event, Australia. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3396851.3403514

1 INTRODUCTION
There are millions of data centers in the world serving the digital demand from personal and industrial users, bringing great changes to our society. This comes at the cost of significant electricity consumption and environmental impacts. Unfortunately, data centers are inefficient in terms of both energy consumption and operation.

Existing cluster resource managers like Yarn and Kubernetes allocate resources based on users’ requests. Since users rarely know their exact resource consumption when submitting their requests, the requested amount can be more or less than the real usage. In fact, our analysis of the Google cluster trace [1] show that the real usage is less than 45% of the requested amount on average.

Due to the utmost importance of the problem, it is not surprising to see lots of efforts have been made to mitigate the gaps between resource usage and requested amount. In particular, oversubscription has been widely used [3]. Specifically, oversubscription assumes demand peaks of different users are rarely collided and thus multiplexing negatively correlated applications on the same server can accommodate more requests and therefore help improve utilization [5]. Despite the benefits, server overloading may occur which results in performance degradation. Another popular approach is to fill the cluster with low-priority jobs, which is used in resource managers such as Kubernetes and Borg. However, using multiple priorities is not a universal solution to all data centers, and it is often challenging to find enough low-priority jobs to fill in the cluster.

In this work, we focus on improving the cluster utilization and maintaining the quality of service (QoS). Given a QoS target, our online resource manager maximizes the cluster utilization while satisfying the QoS target.

Google Trace Analysis. Google cluster trace [1] was collected in May 2011. Figure 1 plots the total usage and request of the cluster. The usage and request are normalized to the capacity of the whole cluster. Both the total requests of CPU and memory are larger than the capacity sometime that means the cluster was over-subscribed. The average total request for CPU is 1.1 while the average total request for memory is 0.9. Memory is sensitive to applications so it is less over-subscribed than CPU. The average usage of CPU (0.43) and memory (0.5) are far under from their capacities 1. However, we believe that the total CPU and memory usages is not the complete proof to the inefficiency of the cluster. So, we need to look into the details of usage and request at machine level.

![Figure 1: [Cluster Analysis] The total usage of cluster is highly underutilized in both CPU (43%) and Memory (50%).](image-url)

2 PROBLEM FORMULATION & SOLUTION APPROACH
There are \(N\) nodes that have the same capacity \(C\). The total of requested resources on node \(i\) is \(R_i\). If the cluster is over-subscribed with the factor \(\theta \geq 1\), \(R_i \leq \theta C\). Meanwhile, the real load (usage) of node \(i\) at time \(t\) is \(L_i < C\). At time \(t\), there are \(J\) pending tasks. Task \(j\) has the constant request \(r_j\) and the future demand \(d_j\). The demand \(d_j\) is unknown prior scheduling.
\( \tilde{d}_j \) varies from the time task \( j \) scheduled till finished. Given demand \( d_j \) and request \( r_j \), the resource allocation for task \( j \) at time \( t \) is \( r_j + s_j \). \( s_j \) is commonly a fair share (FS) or weighted fair share (WFS) of the remaining available resource. We choose weighted fair share (WFS) as a computer often does weighted fair share for its running applications. \( s_j \) can be negative when the real demand is less than its request. Let \( x_{ij} \) be the decision variable for scheduling task \( j \) on node \( i \). If the scheduler decides to place task \( j \) on node \( i \), \( x_{ij} = 1 \). Otherwise \( x_{ij} = 0 \).

**Node Usage based Load Balancing (ULB).** Existing schedulers do the load balancing based on resource requests. Our goal is to balance the actual load across all the nodes by minimizing the maximum utilization \( U \).

\[
ULB : \min_U U \\
\text{s.t. } U \geq \tilde{L}_i + \sum_{j \in J} x_{ij} r_j \quad \forall i \in N \\
U \leq C \\
\sum_{i \in N} x_{ij} = 1 \quad \forall j \in J \\
x_{ij} \in \{0, 1\} \quad \forall i \in N, j \in J
\]

where \( \tilde{L}_i \) is the load information on node \( i \). As the future demand is unknown, we still use the request \( r_j \) in the capacity constraint \( U \geq \tilde{L}_i + \sum_{j \in J} x_{ij} r_j \). We note that \( \tilde{L}_i \) is not the instantaneous load \( L_i \). \( L_i \) should not be simply measured because the load may change overtime and the online optimization problem only captures a single snapshot. If \( L_i \) is too small, we can admit a lot of tasks to the same node but it will result in overloads. If \( L_i \) is too large, it causes low utilization. The key question here is to where to get \( \tilde{L}_i \) before scheduling.

**Load Estimation Penalty.** \( \hat{L}_i \) is an estimated load of node \( i \) at time \( t \). To deal with underestimation or overestimation, we propose using estimation penalty \( P \). The idea is to compute \( \hat{L}_i \) based on the estimated load \( \tilde{L}_i \) and the estimation penalty \( P \) as follows.

\[
\hat{L}_i = P \tilde{L}_i.
\]

While \( \hat{L}_i \) gives us some information about the present and future load, we need to adjust the estimation penalty \( P \) to avoid quality of service (QoS) violations. If \( P \) is too large, \( ULB \) provides guaranteed QoS and low utilization. If \( P \) is too small, \( ULB \) achieves high utilization but violates QoS.

**Quality of Service (QoS).** Let \( q_j \) be the QoS of job \( j \) at time \( t \). \( q_j = f(r_j, d_j, a_j) \) is defined based real resource usage (allocated) \( a_j \) and the resource demand \( d_j \) at time \( t \). \( q_j \) is non-decreasing on \( a_j \). Users requires \( q_j \) greater or equal to the task quality target \( \rho_j \). QoS \( Q \) of the system at time \( t \) is computed as follows.

\[
Q(t) = \frac{1}{|J|} \sum_{j \in J} q_j(t) \geq \rho
\]

where \( I \) is the indicator function.

**Challenges.** \( ULB \) is an integer programming problem and well known as the NP complete problem. It is impossible to find a optimal solution for for scheduling as it needs to be done in subsecond for thousands of nodes and millions of tasks. So, it requires an efficient and fast algorithm. Furthermore, we need to pick the estimation penalty \( P \) before solving \( ULB \). Since we do not have prior-knowledge of task arrival times and task demands in the future, it is challenging to pick the right estimation penalty \( P \).

**Solution.** We break the solution approach into 2 phases: The first one assumes that load estimation is very accurate so we only focus on load balancing. In this step, we can use FIFO (First in first out) or LRF (Large first request).

\[
\text{THEOREM 2.1. If the capacity } C \text{ is infinite, FIFO Scheduling is 2-approximation.}
\]

\[
\text{THEOREM 2.2. If the capacity } C \text{ is infinite and the order of resource requests is the same order of resource demands, LRF scheduling is 4/3-approximation.}
\]

In the second phase, we deal with the errors from load estimation. Flex is the combination of the two phases. Flex is compatible with most task schedulers like Kubernetes, Aurora, or Yarn.

**Evaluation.** We evaluate Flex using Google cluster trace \([1]\) to show that it is better than existing modern schedulers in terms of utilization while maintains the QoS target. We extend Kubernetes Cluster Simulator \([2]\) that is very close to the real Kubernetes code base. There are 4000 nodes. Each node has 64 CPU cores and 128 GB RAM. We compare Flex with Least Fit (LeastFit) and oversubscription (Oversub). LeastFit are featured in Kubernetes and Aurora. Oversub combines oversubscription and LeastFit. The oversubscription factor is 2 and does load balancing like LeastFit does. FlexF and FlexL are the two versions of the proposed algorithm. In Figure 2, FlexF and FlexL are the best in terms of utilization while they are much better than oversubscription in terms of QoS.

3 ACKNOWLEDGEMENTS

This research is supported by NSF grants CNS-1730128, 1919752, 1617698, 1717588 and IITP-2019-11-00783.

REFERENCES

[1] [n.d.]. Google Cluster Traces. https://github.com/google/cluster-data.
[2] [n.d.]. Kubernetes cluster simulator. https://github.com/pfnet-research/k8s-cluster-simulator.
[3] David Breitgand and Amir Epstein. 2012. Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds. In 2012 Proceedings IEEE INFOCOM. IEEE, 2861–2865.
[4] Tan N Le and Zhenhua Liu. 2020. Flex: Closing the Gaps between Usage and Allocation. arxiv (2020).
[5] Xiaqiao Meng, Canturk Isci, Jeffrey Kephart, Li Zhang, Eric Bouillet, and Dimitrios Pendarakis. 2010. Efficient resource provisioning in compute clouds via vm multiplexing. In Proceedings of the 7th international conference on Autonomic computing. ACM, 11–20.

405