Disaggregation in the Cloud with $\mu$Instances and Cirrus

Joao Carreira
UC Berkeley

Abstract

Resource disaggregation can provide significant improvements in the utilization of resources in the datacenter. A Google cluster trace analysis confirms that up to 70% of memory may be recovered with resource disaggregation.

However, resource disaggregation in the cloud is currently unfeasible due to the hardware and network changes required by previously proposed designs. We make the observation that existing cloud offers already provide logical heterogeneity which can be leveraged to match closely the resources demanded by applications. Deploying a system across a heterogeneous set of VMs can lead to higher performance per dollar.

To provide support for deploying systems in logically disaggregated clusters we propose a framework called Cirrus. Cirrus composes small compute-intensive tasks ($\mu$Tasks) with large memory instances to provide the resources required by applications. Cirrus also provides an efficient interface for communication between tasks through distributed memory.

1. Introduction

Disaggregated architectures have received intense and widespread attention by both industry (e.g., HP [7, 11], Intel [3], Huawei [2], and Facebook [1]) and academics [4, 13, 5, 8]. This architecture shift is motivated by recent advances and continued improvement in the bandwidth and latency of networks.

However, existing proposals of physical disaggregation attempt to achieve these benefits by leveraging custom network and memory hardware solutions. For instance, HP The Machine employs a specialized memory fabric. Similarly, a Firebox rack makes use of high-radix optical switches to provide low-latency communication between all resources. This limits the adoption of disaggregation and prevents developers from reaping the benefits of resource disaggregation today.

Surprisingly, we found that logical disaggregation of resources, through the reservation of heterogeneous VMs, can already provide similar benefits. We found that large-scale workloads in the cloud are often better provisioned by a combination of small-CPU and large-memory instances (see Table 1). Instances with a single CPU have a lower cost-per-CPU, while large-memory instances have a lower cost-per-GB-of-RAM. For instance, when deploying a workload with resources \{CPU: 100, RAM: 100\} in Amazon AWS, a deployment based on medium-sized $m4.large$ instances costs $5.5/\text{hour}$ while a combination of small CPU instances (e.g., $t2.nano$ or $lambda$) with large memory instances (e.g., $r4.large$) costs $1.43/\text{hour}$ and $1.1/\text{hour}$, respectively (see Figures 1).

In this paper we propose Cirrus, a framework that provides application-level disaggregation of resources in existing cloud environments through the use of software abstractions. Cirrus combines the use of small compute-intensive instances (which we call $\mu$Instances) with large memory instances. This approach allows better utilization of cloud resources and lower cost with similar performance by leveraging the heterogeneity of cloud instances.

We used Cirrus to build distributed machine learning algorithms based on the distributed parameter server model.
2. Design and Implementation

To address the challenges of supporting logical disaggregation we propose a framework called Cirrus (see Figures 2 and 3). Cirrus provides support for deploying and connecting resource-specific application tasks in the cloud. First, it provides an interface developers can use to build tasks. Developers can annotate these tasks with the type and amount of their resource demands. These annotations are then used to deploy the tasks into cost-efficient VMs. Lastly, Cirrus provides an efficient software interface with caching and prefetching on top of an optimized network layer for efficiently accessing data stored in storage tasks.

Unlike CPU and GPU tasks that are programmed by developers, memory and storage tasks are provided by Cirrus out of the box. These tasks provide an uniform object store (put/get) interface that can be used by applications to store working data. The interface is highly optimized to serve as a fast remote memory interface. On top of the put/get interface, Cirrus provides a Cache Manager and an Iterator interface. The Cache Manager caches objects in the local memory of instances to avoid remote accesses. It also allows custom eviction and prefetching policies that can be used by developers who know the data access patterns of their applications best.

On top of the Cache Manager, an Iterator interface allows applications to iterate over sets of objects. This provides good performance for typical sequential access patterns. The iterator can also be configured with random and application-specific iterator policies. We believe this can be useful for applications with well known access patterns (e.g., graph algorithms).

Cirrus is built in roughly 10K lines of C++14 code. The object store components are designed and developed to be a substitute for the limited amount of local memory of instances, and thus are highly optimized to provide as low overhead as possible. The object store interface supports TCP and RDMA transports so that Cirrus can be used both with commodity and specialized network interconnects.
References


