Kraken: Towards Elastic Performance Guarantees in Multi-tenant Data Centers

Carlo Fuerst[1], Stefan Schmid[1,2], Lalith Suresh[1], Paolo Costa[3]
[1] TU Berlin, Germany; [2] T-Labs, Berlin, Germany; [3] Microsoft Research, UK
{carlo,stefan,lalith}@inet.tu-berlin.de; paolo.costa@microsoft.com

ABSTRACT

It is well-known that without strict network bandwidth guarantees, application performance in multi-tenant cloud environments is unpredictable. While recently proposed systems support explicit bandwidth reservation mechanisms, they require the resource schedules to be announced *ahead of time*. We argue that this is not practical in today's cloud environments, where application demands are inherently unpredictable, e.g., due to stragglers. We in this paper present KRAKEN, a system that allows tenants to dynamically request and *update* minimum resource guarantees for both network bandwidth and compute resources *at runtime*. Unlike previous work, Kraken does not require prior knowledge about the resource needs of the tenants' applications but allows tenant to modify their reservation at runtime. Kraken achieves this through an *online resource reservation scheme*, and by optimally embedding and reconfiguring virtual networks.

1. INTRODUCTION

Cloud-based applications, including MapReduce and scale-out databases, generate significant amount of network traffic and a considerable fraction of their runtime is due to network activity. Unfortunately, as reported in previous studies [1], in existing cloud infrastructures the bandwidth available to the tenants varies significantly over time, even within the same day. Given the time spent in network activity by these applications, this variability has a non-negligible impact on the application performance, which makes it impossible for tenants to accurately estimate the execution time in advance [3].

Several systems have been proposed which leverage admission control and support explicit bandwidth reservations to overcome this problem. [1, 4] Many of these systems offer *virtual cluster* abstractions, which provides the tenants with the illusion of having their own dedicated network: A virtual cluster offers the tenant the illusion for all her *Compute Units (CUs)* to be attached to a single non-oversubscribed switch with a minimum bandwidth b guaranteed. A virtual cluster $\mathrm{VC}(n, b)$ has two parameters: n, the number of (identical) CUs in the cluster, and b, the bandwidth reservation from each CU to the virtual switch.

SIGMETRICS'15, June 15-19, 2015, Portland, OR, USA.
ACM 978-1-4503-3486-0/15/06.
http://dx.doi.org/10.1145/2745844.2745879 .

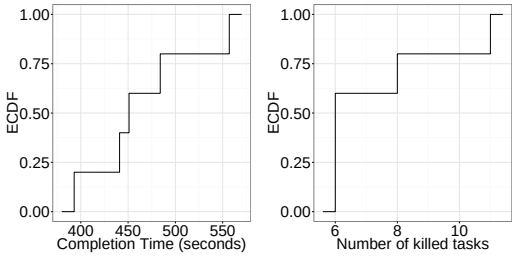

Figure 1: Execution unpredictability—Completion times of jobs in the presence of speculative execution (*left*) and the number of speculated tasks (*right*).

However, *all* existing solutions providing absolute bandwidth guarantees are based on *offline reservations schemes* [1, 2, 4]: they require that tenants announce the entire resource reservation schedule *ahead of time*, i.e., at job submission time.

We argue that in most cases it is very hard to accurately estimate application resource needs ahead of time, rendering offline reservation schemes inadequate. To highlight this point, we demonstrate that even with the same workload and a dataset of the same size being re-executed, it is difficult to predict how a job progresses over time. Concretely, we conduct an "idealized" experiment wherein we run a Hadoop cluster in our OpenStack-based testbed; environments such as Amazon EC2 are far more noisy. We use five physical servers (8 CPU cores) with one virtual machine each. Each virtual machine is allocated 4 virtual cores. Each node of the Hadoop cluster is mapped to a virtual machine each (one master, four slaves). The Hadoop workload consists of a TeraSort job. We repeat the experiment five times with speculative execution enabled. Figure 1 (*left*) indicates the variance in job completion times across the runs: a range of 150 seconds. This observation is also supported by Figure 1 (*right*) which shows the variable number of straggling tasks speculatively re-executed by the Hadoop cluster.

2. KRAKEN

Kraken introduces virtual clusters whose resource guarantees can be adjusted over time, in an online fashion. Since the objectives of the execution (e.g., deadlines) depend on the context and the need of the tenant, we argue that the tenant itself should be responsible for the specific scheduling strategy, and Kraken simply exposes an interface which allows to (1) upgrade a virtual cluster $\mathrm{VC}(n,b)$ consisting of n CUs and with a bandwidth guarantee b, both in *size* (i.e., number of CUs) as well as in the minimum *bandwidth*, that is, to a virtual cluster with $x \geq 0$ more nodes and a factor $\delta \geq 1$ more bandwidth, i.e., to $\mathrm{VC}(n + x, b \cdot \delta)$; (2) downgrade

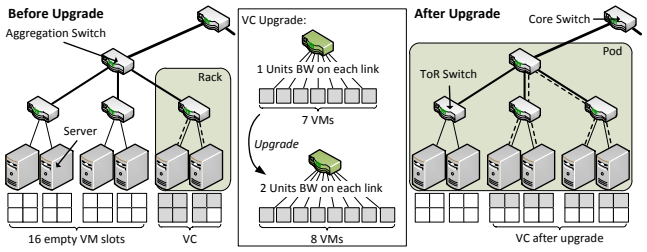

Figure 2: Example: upgrade of virtual cluster requiring migration.

a virtual cluster in both size and bandwidth; (3) or a combination of both (e.g., upgrade size and downgrade bandwidth).

How to support such reconfigurations is also an algorithmic problem. We would like to avoid or at least minimize migrations in order to satisfy a reconfiguration request; moreover, the resulting embeddings should have small network *footprints*, in the sense that CUs are embedded close to each other and unnecessary bandwidth reservations (on substrate links) are avoided.

Example. To illustrate both the model and the challenge, let us consider an example. Figure 2 (*left*) shows a part of a fat-tree data center topology, i.e., a single pod consisting of three racks with two servers each; each server has 4 CU slots. We assume that the uplinks of the servers have a capacity of 4 units and the fat-tree provides full bisection bandwidth, resulting in a capacity of 8 units on the ToR switches' uplinks and a capacity of 24 units on the links between the aggregation switches and the core switch. On the right most rack, currently a virtual cluster VC is embedded; the dashed line indicates the path along with bandwidth is reserved to connect the CUs. At some point, VC is upgraded, from VC(7,1) to VC(8,2), see Figure 2 (*middle*).

How can this request be implemented? Theoretically, the right server in the rack still has a free CU slot which could be used to accommodate the additional CU; however, doubling the bandwidth reservations for each of the CUs will violate the bandwidth capacities on the uplinks of the servers. Hence it becomes necessary to distribute the CUs in the substrate, in order to reduce the bandwidth utilization of the uplinks of the two servers. Thus, in this scenario, some CUs need to be migrated to satisfy the request. Figure 2 (*right*) shows a solution: the resulting embedding is valid.

Theoretical Contribution. Kraken is based on the observation that by first fixing the logical switch of the virtual cluster, optimal tradeoffs between small footprints and reconfiguration costs can be computed efficiently, using dynamic programming (namely bottom-up, from the substrate-tree leaves to the core switch). Since also the number of possible logical switch locations is small (linear in the substrate size), Kraken is fast.

THEOREM 2.1. *Kraken efficiently computes provably optimal virtual cluster embeddings and minimizes reconfiguration costs.*

Previous Work: Static Embeddings only and Suboptimal Footprints. Previous systems such as Oktopus [1] and Proteus [4] do not support online reconfigurations and migrations. Moreover, the static embeddings computed by these systems can be suboptimal (large footprints): In the case of Oktopus, a small virtual cluster which could be hosted by a single server (Kraken footprint: 0) may be embedded across multiple servers (footprint > 1). In the case of Proteus, the ratio of the optimal footprint computed by Kraken and the footprint by Proteus can be as high as $n/3$: such an example can be constructed by exploiting the fact that Proteus will only consider cross-pod embeddings if a request cannot be fit in a single pod. Thus, in case $n-1$ slots are available on a single server in one pod and n times one slot is available on servers in another pod, the Proteus footprint is $2n$ while the footprint of Kraken is 6.

Preliminary Evaluation. Hadoop-YARN is an interesting application for Kraken, as the MapReduce tasks inform the Application Master about the progress of a task periodically. This information is used, for instance, to speculate tasks that are straggling. To demonstrate a basic version of Kraken (without support for elastic computations or migrations), we implemented a simple controller that runs inside a virtual machine and uses the Linux `tc` utility in order to make bandwidth reservations. We then instrumented the Hadoop source code such that tasks inform the controller prior to executing a shuffle. If there is spare bandwidth to be allocated, the controller increases the corresponding endpoint's bandwidth reservation. Once the shuffle is completed, the Hadoop framework informs the controller of the same in order to release its reservations. (Migrations are not supported yet.)

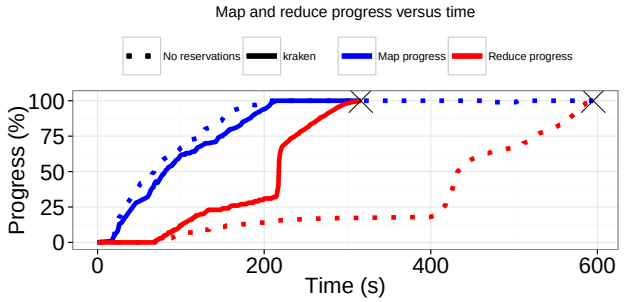

Figure 3: Map and reduce progress of a TeraSort job in best effort conditions (*dotted*) and when using Kraken (*solid lines*). The X marks indicate the point of job completion.

Using this framework, we executed the TeraSort benchmark against a dataset size of 100 million 100 byte records (a total of ~10 GB of data). We co-locate the Hadoop datanodes against a set of VMs that generate UDP flows on the network using `iperf`. The UDP flows are set to last 400 seconds at a throughput of 600 Mbps and thus significantly stresses the underlying 1 Gbps network. Hadoop is initially allocated only 100 Mbps of bandwidth, but requests additional bandwidth when shuffles are to be executed (with the UDP tenant given best-effort service, and left to consume the remaining bandwidth throughout). Figure 3 indicates the map and reduce progress with and without Kraken's online reservations. Without online reservations, the UDP tenant interferes heavily with Hadoop's network usage, thus prolonging the TeraSort job until the UDP flows terminate. With online reservations however, Hadoop requests bandwidth when it needs it, leading to just under 300 seconds of improvement in job completion time.

Research Funding. BMBF Software Campus grant 01IS12056, Bigfoot FP7-ICT-317858, G.I.F. No I-1245-407.6/2014.

3. REFERENCES

[1] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *Proc. ACM SIGCOMM*, 2011.

[2] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. SecondNet: A data center network virtualization architecture with bandwidth guarantees. In *Proc. ACM CoNEXT*, 2010.

[3] J. C. Mogul and L. Popa. What we talk about when we talk about cloud network performance. *SIGCOMM CCR*, 42(5):44–48, 2012.

[4] D. Xie, N. Ding, Y. C. Hu, and R. Kompella. The only constant is change: incorporating time-varying network reservations in data centers. In *Proc. ACM SIGCOMM*, 2012.