Opening Up Black Box Networks with CloudTalk

Costin Raiciu Mihail Ionescu Dragoș Niculescu
University Politehnica of Bucharest

Abstract

Optimizing distributed applications in the cloud requires network topology information, yet this information is kept confidential by the cloud providers. Today, applications can infer network properties and optimize accordingly but this is costly to get right. The cloud can optimize the network via load balancing, but the scope is limited to moving traffic between the available paths.

In this paper we challenge this status quo. We show that network topology information is not confidential in the first place by conducting a study of Amazon's EC2 topology, and we show how such information can have a big impact in optimizing a web search application.

We propose that applications and the network should break the silence and communicate to allow better optimizations that will benefit both parties. To this end we design CloudTalk, a very simple language that allows apps to express traffic patterns that are then ranked by the network. The ranking helps application pick the right way to implement its tasks. We provide a proof-of-concept implementation of CloudTalk showing that it is expressive enough to capture many traffic patterns and that it is feasible to use in practice.

1 Introduction

Many organizations are running large-scale distributed applications in the cloud, benefiting from the abundance of resources and the attractive pay-per-use billing model. Cloud users want their apps to finish as quickly as possible, and they should be able to use the many optimizations that have been proposed to this end [8, 4, 10]. Optimizations often require detailed knowledge of the cloud topology and its resources. In particular, network topology information is crucial to getting good performance as the network is often the bottleneck. Meanwhile, the datacenter operators keep topology information confidential.

This status quo is suboptimal: application optimizations either ignore the network or profile it continuously to get accurate information. The former results in disgruntled customers and low network utilization and the latter wastes traffic. Network-only optimizations are limited to load-balancing traffic between available paths. We are seemingly stuck: optimizing distributed applications requires in-depth topology knowledge that providers are understandably reluctant to give away.

In this paper we challenge this status quo. First, we seek to understand just how private network topology information is in practice by conducting a measurement study on Amazon's EC2 Infrastructure-as-a-Service cloud. We find that hiding the network topology is rather difficult: our measurements give a clear blueprint of the EC2 network topology which resembles VL2 [7] (see Section 2). To understand whether topology information does help optimizing applications, we deploy a distributed web-search application on 100 servers. We run several configurations of this application, one of which has been optimized using the inferred topology. The results show that substantial benefits can be had when topology information is used (Section 3).

Our practical experiments show that the topology information cannot be hidden, and that optimizing applications requires such information. How should networks and applications communicate to help optimizations? One way transfer of information from the network to the applications is difficult to get right: either the information accurately describes the topology but violates the operator's confidentiality requirements, or the information is somehow anonymized and certain optimizations are not possible anymore.

In Section 5 we propose an alternative where both applications and the network provide information. We design a simple language called CloudTalk that allows applications to succinctly specify network tasks—each containing related network transfers—and submit them to the network. The network responds with task finish time estimates. These are used for optimizations: applications will choose between different equivalent ways of doing the same computation.

We have implemented a proof-of-concept implementation of CloudTalk and applied it to optimize web search and MapReduce apps. Initial results show that CloudTalk is expressive enough to optimize interesting applications and feasible to implement in practice.

2 Inferring Cloud Topology Information

Cloud providers such as Amazon EC2 or Microsoft Azure do not disclose any information regarding network topology. Nevertheless, a great deal of information can still be inferred by running measurements on cloud instances. We analyzed the topology of Amazon EC2's us-east-1d datacenter by acquiring many "instances" and using well known network diagnosis tools such as ping, traceroute and iperf. We only describe our main findings here due to space constraints.

Amazon EC2 instances are virtual machines running on top of Xen [3] hypervisor. On each physical machine there is a special VM called Dom0 and many tenant VMs. Dom0 controls the tenant VMs's access to the network and acts as an IP hop and the default router for the guest machines. Because of this we can only infer L3 topology information. We ran ping and traceroute between all our instances and identified several situations:

1. The instances (or Virtual Machines, VMs) are on the same physical machine. In this case, there is only one hop between the two VMs, the packets go from VM_1 – Dom0 – VM_2. The IP addresses of the VMs are usually very close to each other, in the same /24 address space. Using iperf we find that the available bandwidth between such VMs is pretty high at around 4Gb/s.

2. The VMs are in the same rack. We assume the existence of a top of the rack switch (ToR), but this only functions at L2. There are two hops between the VMs: VM_1 – $Dom0_1$ – $Dom0_2$ – VM_2. The IP addresses of the VMs are in the same /24 address space. The available bandwidth is 1Gb/s.

3. The VMs are in the same subnet. We found out that a subnet contains machines with addresses of the form 10.n.x.x and 10.n+1.x.x; for example 10.194.x.x and 10.195.x.x will be in the same subnet. In this case, there are three hops: VM_1 – $Dom0_1$ – EdgeRouter – $Dom0_2$ – VM_2. We run the experiments 100 times between each pair of machines and we found out that there are actually two edge routers for each subnet, and it seems that *Dom*0 (or, less likely, the ToR switch) is load balancing traffic to the routers dynamically. The available bandwidth is 1Gb/s.

4. The VMs are in different subnets. In this case, there are five intermediate hops: VM_1 – $Dom0_1$ – EdgeRouter – CoreRouter – EdgeRouter – $Dom0_2$ – VM_2. Running the experiments multiple times, we found that there are two groups of core routers, each of them connecting all edge routers from the different subnets, and providing full redundancy. The available bandwidth is still 1Gb/s, confirming anecdotal evidence that EC2 has full bisection bandwidth.

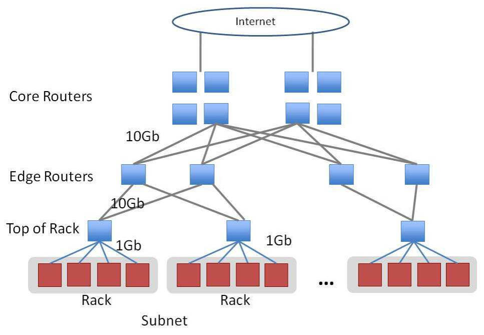

Figure 1: EC2 US-EAST Availability Zone D Topology as inferred by measurement in December 2011

An interesting situation surfaced during these tests. The data from traceroute and ping -R was not quite in sync, showing different IP addresses for the same routes. We hypothesized that the actual routers were the same machines, but the addresses reported were coming from different router interfaces. We proved this true by employing a well-known technique for interface disambiguation [2]. The main idea is that a router, when transmitting an ICMP message back, will use the source address of the outbound interface on which the packet is sent. Therefore, one can transmit a UDP packet to an unused port at each of the interfaces I_1 and I_2 (suspected to be on the same physical machine) from the same source. If the returned ICMP packet comes from the same I_3, it means that I_1 and I_2 belong to the same router.

We present in Figure 1 the topology for EC2's us-east-1d datacenter as resulting from our experiments. While we cannot be absolutely sure that this is 100% accurate, it correctly explains all behavior we noticed during our tests, as well as anecdotal information gathered from various informal meetings with people from Amazon. The topology is very similar to the VL2 topology [7].

3 Optimizing Web Search

Web search is a classical distributed application which uses a "scatter-gather" workflow. Nodes are organized in a hierarchical structure. The query is sent by the frontend towards the leaves, while the results go in the opposite direction. The architecture we used for our experiments has two levels of aggregators and is shown in Figure 2. Each aggregator has around 50 nodes underneath.

For our implementation, we used Apache Solr on top of the Apache Tomcat web server. This combination is considered by many as the state of the art in open source search engines. Solr naturally supports aggregation on multiple levels using the concept of *sharding*,

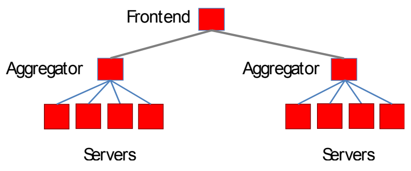

Figure 2: Web Search architecture with two levels of aggregation

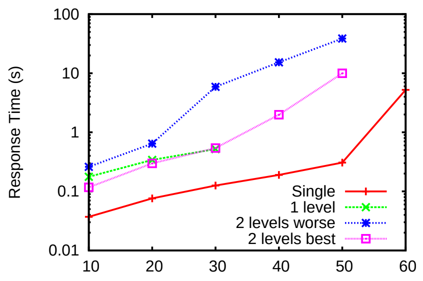

Figure 3: Web search performance

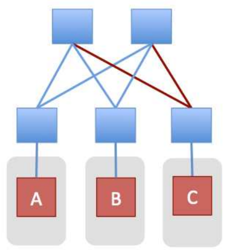

Figure 4: Simple MapReduce optimization: who should run the two reduce jobs?

where an instance can be configured to return aggregated results from its own data together with results from other servers. Each machine is a High CPU medium instance, with 1.7GB of RAM and 5 EC2 computing units hosting the content of roughly 5 Million indexed URLs for a total of around 4GB of data (the web pages are from a 2.5TB snapshot of the .uk domain taken in 2009).

We measured the performance of the system in multiple configurations and present the results in Figure 3. First, we measure the raw performance of one machine searching its part of the index — this the baseline for the distributed measurements. Second, we measured with only one aggregator contacting all 100 servers. When traffic was low, the system was behaving correctly but with obviously higher delays compared to the single-machine baseline.

Once the traffic exceeds 35 queries per second (qps), the aggregator software started crashing. We believe that the effect is due to TCP incast[13]: each server sends bursts of packets containing query replies to the corresponding aggregator via a single TCP connection. The limited network buffer available on the aggregator's port in the ToR switch is overrun, losing many packets. Each TCP connection has few packets in flight so fast retransmit is not triggered, resulting in a costly timeout. Meanwhile, unserviced requests pile-up at the frontend until the server runs out of memory and crashes.

The last two tests were done in a configuration similar with the one presented in Figure 2. The only difference among the two tests was the location of the aggregators. We kept the frontend and the nodes constant and we varied the location of the aggregators, using the inferred topology information presented in the previous section.

For the first case (called "worst"), we placed the aggregators such that each aggregator and its corresponding nodes were in different subnets. For the second case (called the best), each aggregator was placed in such a way to to maximize the number of servers close to the aggregator. For the actual machines we had, this meant that one aggregator had 14 machines in the same rack, 15 in the same subnet and 18 in different subnets, while the other had 20 machines in the same rack, 10 machines in the same subnet and 18 in different subnets.

The results show that the effects of incast are smaller compared to the single aggregator scenario. Further, when traffic increases, the performance for the "best" scenario gets substantially better than the "worst" case, even by a factor of 3 to 1. We conjecture this is because the RTT distribution is more varied in the best case; we confirm this effect via simulation in Section 6.

4 The Need for Cloud Talk

We have shown that "confidential" topology information can be inferred and is key to getting good performance in datacenters. While applications and networks could go on like this, the status quo is suboptimal for all parties. Take the example shown in Figure 4 of a scheduler that must schedule two reduce jobs and it can choose between machines A, B and C that have just finished running map jobs. If asked, the network would advice it is best run the reduce jobs on A and B because C's downlinks are congested. In practice, the application will start the reduce jobs on two random machines, and perhaps reschedule if one lands on C and becomes a straggler; however this takes time and wastes resources. Once the scheduler assigns the reduce job, the network can only load balance flows onto its core links for small improvements.

It therefore seems inevitable that the network and the applications should collaborate to reap mutual benefits. Although there are many ways to design the information flow between the network and its applications, solutions must provide a number of desirable properties including: **expressiveness** to capture complex application data patterns, **efficiency** to allow constant use, **dynamicity** to allow optimizations on short timescales and finally **reduced information leakage**.

The most obvious solution is to have the datacenter operator provide each requesting application the real network topology together with buffer sizes. This violates the confidentiality requirement, and does not capture any dynamic information the network may have, for instance the list of congested links or current buffer usage.

Another solution is to adopt ideas from IETF ALTO (application-layer traffic optimization) Working Group [12]. ALTO aims to help network operators and endpoints to communicate to solve the following tussle: BitTorrent clients draw most traffic from the best peers, but in the process they create a lot of costly upstream traffic for their ISPs. ALTO servers run by the operator provide requesting applications with a network map and a cost map. The network map is essentially a clustering of IP addresses into groups called PIDs, and the clustering is performed by the operator according to its own routing policy. The cost map provides routing costs between PIDs. The applications use this information to select the most desirable peers from the point of view of the network and the application.

The ALTO model offers better confidentiality than the solution above, but it is not as expressive as we need it to be in a datacenter: it does not capture costs in many-to-one (e.g. example above) or many-to-many cases, and does not capture dynamic information.

5 CloudTalk Language

We propose a simple solution where applications explicitly ask the network for advice by providing a request which includes a number of equivalent network tasks. The network ranks these alternative tasks using its detailed knowledge of the topology and possibly the current traffic load. The application will then use the most efficient task as told by the network. This type of interaction relieves the network from having to provide potentially verbose, confidential data. The application provides little confidential information beyond what the network could already find by looking at the traffic.

A task is a set of interrelated network transfers specified by the application; the task finishes when the last transfer of that task finishes. For each individual transfer, the application specifies a task-unique *identifier*, *the source*, *the destination*, the *size* and the *start time* of the transfer. All times are given as wall-clock seconds and are relative to the start of the task. Additionally, start times can refer to the finish times of other transfers.

Let's consider an application wishing to run a scatter-gather task. The data can be queried from hosts A or B, and the data resides on hosts C and D. Which of A and B should be used? The application will create two tasks, one running scatter-gather from host A and one from host B. The description of Task A is:

```
ID From To  Size  Start
1   A    C  1000   0
2   A    D  1000   0
3   C    A  10000  finish(1)
4   D    A  10000  finish(2)
```

Task B's description is similar. The application submits both task descriptions to the network, and the network ranks them according to their performance.

This description model is very simple yet it can capture most of the traffic patterns we care about in practice. One-to-one communication is trivial to express, and many-to-one or many-to-many interactions can be easily described with tasks similar to the one above. The strength of the model stems from the fact that it includes time explicitly: apps can tell the network which flows are parallel and which depend on other flows. For the latter part, the $finish(ID)$ operator is used.

The start time of a task is an arithmetic expression using scalars and other $finish$ operators, as well as *max*/*min* functions. This allows us to express synchronization barriers, such as those seen at the end of the shuffle phase of MapReduce computations. To allow more concise representations, traffic patterns that include multiple sources or destinations can be specified as long as all transfers send the same amount of bytes. In such cases identifiers are assigned sequentially.

The model also contains the $bandwidth$ operator that allows the applications specify more complex communication patterns. One example is daisy-chaining that is used by distributed file systems to store data simultaneously on a chain of hosts, with all intermediary hosts sending data out as soon as it is received. A model of daisy chaining can leverage the $bandwidth$ operator to limit the throughput on link $i+1$ in the chain to be always smaller than or equal to the throughput of the predecessor link i. To decide replica placement, a distributed filesystem can use the following task description:

```
ID From To  Size  Start
1   A    B  64MB   0
2   B    C  64MB   0
Constraint: bandwidth(2)<=bandwidth(1)
```

6 Optimizing Clouds with CloudTalk

Implementing CloudTalk requires changes to both applications and the network. Changing applications is notoriously difficult (because of their sheer numbers) but we believe that this is feasible in datacenters where a few key platform applications—such as distributed computation (e.g. Hadoop), distributed filesystems (e.g. HDFS) and search—account for the majority of uses. All these applications feature one or more scheduler machines that decide how to run certain tasks; to support CloudTalk we need to change the schedulers to query for and use network-provided optimization hints. As there is only a handful of widely-used schedulers, we believe modifying these to support CloudTalk is straightforward. We intend to implement this in our future work.

The network can implement CloudTalk by providing one or a few servers that know the network topology (and maybe the instantaneous traffic load) and answer application queries. To answer queries, the network servers can use packet and flow-level simulation and even linear programming. Given an application request, the network (servers) can simulate servicing the request and then use the results to rank the alternatives provided by the application. We implemented a proof-of-concept implementation of CloudTalk as an extension to the *htsim* simulator. *htsim* is a packet-level simulator that can scale to moderately sized datacenters containing hundreds to a few thousand nodes. The advantages of packet-level simulation include the ability to provide accurate results for both long-lived transfers as well as incast situations. On the downside, packet-level simulation time scales linearly at best with the size of the transfers—this may prove costly for analyzing MapReduce tasks or large filesystem reads and writes.

To understand CloudTalk's feasibility we used *htsim* to simulate a modified VL2 topology containing 1200 servers. The simulated topology models the topology we inferred in practice. Host-to-switch links are gigabit, while switch-to-switch links are 10 gigabit.

Our first experiment emulates the search application described in section 3. We used the inferred topology information to place the 100 servers in appropriate racks and subnets in the simulated topology. We then generate three CloudTalk task descriptions describing the network traffic patterns of the query application with one aggregator, two aggregators (local) and non-local.

When the simulator receives a task description it parses it, instantiates the desired flows and then simulates the transfers in an otherwise idle network. Each task is simulated ten times to capture non-determinism induced by ECMP routing in the network. The results given in the table below give estimated average delays in milliseconds. We provide results for three configurations of the network corresponding to different switch port buffer sizes for the host links. Simulating one task 10 times takes around 1.5s of wall-clock time.

	Buffer Size (pkts)		
	30	50	100
One aggregator	1514	1045	506
Two agg. (worst)	832	541	421
Two agg. (best)	812	423	420

The task model assumes a single query packet is sent out and ten packets are returned by each server immediately after the query is received (we make the simplifying assumption that all servers process a request instantly). To mimic the Linux stack we used, htsim's TCP's retransmission timer is lower bounded at 200ms. The simulation results qualitatively agree with our experimental findings - the best setup is the one where two local aggregators are used. Further, the simulations confirm that incast is the root cause of the increased delay observed. It is however unclear why the best setup behaves significantly better than the case where aggregators are farther away from the servers—the reason may be the fact that the local setup has a wider range of RTTs from the servers to the aggregator: a third of hops will be two links away from the server, another third will be 4 links away, and the rest will be 6 links away. In the worst case all servers are 6 hops away from their aggregator and this will increase the contention for the aggregators' buffer.

Our second experiment analyzes the map phase of MapReduce computation in the same cluster when there are non-local map tasks to run. Here the network is asked to rank two file transfers of 64MB from their source to the possible mappers. *htsim* answers this task in 1.2s, which is due to the sheer number of packets it needs to simulate (simulation time scales linearly with the number of packets simulated).

7 Discussion

Assumptions CloudTalk assumes applications will know the amount of bytes they want to transfer and the task dependencies. Many applications already have this information: a GFS client will know the chunk metadata and the servers it wants to transfer from [6]. A MapReduce scheduler has data size and placement information for map inputs and outputs as well as reduce outputs.

Further, CloudTalk assumes that the network can give accurate answers to queries. Exact answers may be challenging to give because of dynamic traffic outside the cloud provider's control, as generated by other tenants and by software not using CloudTalk. To alleviate this problem the network provider could query its switches to figure out the large flows utilizing its links, as shown by Hedera [1] or DevoFlow [5]. This information would then be used when answering queries.

Limitations By default, the CloudTalk model assumes all traffic is TCP and that it is not application bound. Bandwidth constraints can be used to specify known application bounds as well as constant-bit rate traffic; other congestion control algorithms are not yet supported.

The language does not capture traffic burstiness within the same connection: video traffic follows this pattern, where periods of high throughput that fill the client's play-out buffer alternate with idle periods. Finally, CloudTalk does not express computational bottlenecks which can easily dominate a job's finish time.

CloudTalk optimizations involve at least a round-trip time where the application queries the network and receives an answer. Even assuming the network replies instantaneously, waiting an additional RTT for every query is unacceptable for web-search. In such cases, the application can still use CloudTalk to optimize the aggregation tree but only at deployment time. As a consequence, such optimizations can only rely on static topology information.

Scalability For CloudTalk to be practical, the network needs to respond quickly to application requests. Our prototype shows that an existing packet-level simulator answers certain queries reasonably quickly without any additional optimizations. However, packet-level simulation scales poorly with the number of bytes transferred, and this is a problem for large scale data processing jobs.

To reduce this overhead implementations could make simplifying assumptions such as scaling down the sizes of the simulated transfers by one or a few orders of magnitude. This, however, will impact the accuracy. Choosing an appropriate tradeoff between response time and response accuracy requires further study.

An alternative to packet-level simulation are flow-level simulations that scale with the number of flows simulated and are invariant of transfer sizes. The problem with such analytical solutions is that they only work well when the number of flows is constant. Adapting analytical schemes to deal with dynamic flow arrivals, and analyzing their feasibility is subject of our future work.

8 Related Work

Many recent works use topology information to optimize distributed applications in the datacenter. Purlieus[10] uses this knowledge to instantiate virtual machines at appropriate places in the cluster such as to reduce network bottlenecks for different types of load: map intensive, shuffle intensive, or reduce intensive. Orchestra [4] proposes the use of transfer controllers and global coordination to achieve scheduling across jobs, and across the entire datacenter. They show that scheduling schemes based on maxmin sharing between flows lead to 1.5x reduction in the shuffle component of the MapReduce.

Mesos [8] proposes a two-level scheduler hierarchy for optimizing resource usage among different frameworks. Mesos shares our goal of making the datacenter and the applications communicate: the datacenter makes resource offers and applications choose among available ones. Mesos focuses on CPU and memory resources and assumes applications know which resources they want. In contrast, CloudTalk targets the network and assumes applications do not know what the topology is.

Optimizations such as Hedera [1] or MPTCP [11] have been proposed to alleviate the effects of collisions arising from random hashing of flows to paths with ECMP [9].

9 Conclusions

Dialogue between the network and the applications has been acknowledged as the way to solve conflicting goals of P2P users and network providers. We argue that a similar dialogue is necessary in the datacenter. Providers want to keep their network topology a trade secret, while the users need it to optimize their applications. Infrastructure cannot be completely hidden from the users, and we show that even with limited knowledge of the topology it is possible to gain significantly: a web search application on EC2 can be as much as 3 times faster with proper placement of the aggregators.

We propose CloudTalk, a language designed to allow users to describe their network tasks such that the network can accurately estimate their completion time, and this information then enables application optimizations. Our simulations show that CloudTalk is both expressive and feasible in practice: tasks describing web search are ranked correctly in under 1.5s for a 1200 node network.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable feedback. C. Raiciu has been partially supported by the CHANGE research project, financed by the European Comission through its FP7 programme. M. Ionescu and D. Niculescu have been partially supported by the project POSDRU/89/1.5/S/62557.

References

[1] Mohammad Al-Fares et al. Hedera: Dynamic flow scheduling for data center networks. In *Proc. Usenix NSDI 2010*.

[2] Paul Barford et al. On the marginal utility of network topology measurements. IMW '01, ACM, pp. 5–17.

[3] Paul Barham et al. Xen and the art of virtualization. SOSP '03, ACM, pp. 164–177.

[4] Mosharaf Chowdhury et al. Managing data transfers in computer clusters with orchestra. SIGCOMM '11, ACM.

[5] Andrew R. Curtis et al. Devoflow: scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 conference* (New York, NY, USA, 2011), SIGCOMM '11, ACM, pp. 254–265.

[6] Sanjay Ghemawat et al. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2003), SOSP '03, ACM, pp. 29–43.

[7] Albert Greenberg el al. VL2: a scalable and flexible data center network. In *Proc. ACM Sigcomm 2009*.

[8] Benjamin Hindman et al. Mesos: a platform for fine-grained resource sharing in the data center. NSDI'11, USENIX Association.

[9] C. Hopps. RFC 2992: Analysis of an equal-cost multi-path algorithm, Dec. 2000. http://tools.ietf.org/html/rfc2992.

[10] Balaji Palanisamy et al. Purlieus: locality-aware resource allocation for mapreduce in a cloud. SC '11, ACM, pp. 58:1–58:11.

[11] Costin Raiciu et al. Improving datacenter performance and robustness with Multipath TCP. In *Proc. ACM SIGCOMM 2011*.

[12] J. Seedorf and E. Burger. RFC 5693: Application-layer traffic optimization (ALTO) problem statement, Oct. 2009. http://tools.ietf.org/html/rfc5693.

[13] Haitao Wu et al. ICTCP: Incast congestion control for TCP in data center networks. Co-NEXT '10, ACM, pp. 13:1–13:12.