Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2013-003 February 12, 2013

A Plan for Optimizing Network-Intensive Cloud Applications

Katrina LaCurts, Shuo Deng, and Hari Balakrishnan

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

A Plan for Optimizing Network-Intensive Cloud Applications

Katrina LaCurts, Shuo Deng, and Hari Balakrishnan
MIT Computer Science and Artificial Intelligence Lab, Cambridge, MA
{katrina, shuodeng, hari}@csail.mit.edu

ABSTRACT

A significant and growing number of applications deployed on cloud infrastructures are *network-intensive*. These applications are frequently bottlenecked by the speed of network connections between the machines on which they are deployed. Due to the complexity and size of cloud networks, such applications often run slowly or have unpredictable completion times and/or throughput, both of which can result in increased cost to the customer. In this paper, we argue that cloud customers should be able to express the demands and objectives of their applications. We outline an architecture that allows for this type of expression, and distributes applications within the cloud network such that the application's objectives are met. We discuss some of the key questions that need to be addressed to implement the architecture, as well as the interactions between optimizations done by clients and by cloud providers. We also present preliminary results that indicate that these types of systems are feasible and improve performance.

1. INTRODUCTION

In recent years, datacenters and cloud computing infrastructures have become popular platforms for running network-intensive applications. No longer simply a platform for scaling web servers, these infrastructures must now support applications that transfer large amounts of data between machines in the cloud (see `https://aws.amazon.com/hpc-applications/` for some examples). Unfortunately, networks in datacenters and public cloud infrastructures are complex, and path rates can vary significantly due to cross traffic, differences in link qualities, oversubscription, etc. [4]. Customers running network-intensive applications on these networks can see slow and unpredictable run times, which results in increased cost to the customer, and makes it difficult to plan.

We posit that many of these problems could be mitigated by doing a better job of distributing applications across machines in the cloud network. In today's clouds, customers are given access to a set of machines on the network, presumably optimized for some purpose (for instance, the cloud provider may provide machines near each other). Once the customer has their set of machines, they may distribute their application arbitrarily across them; we refer to this distribution as a *placement* of the application. Unfortunately, there is no interface in place for the customer to express the requirements of their application to the provider; the cloud provider is agnostic to the application when assigning the customer a set of machines, and their machine assignment may not be aligned with the needs of the application. Moreover, once assigned a set of machines, there is no service available for the customer to distribute their application across those machines in an intelligent manner, perhaps to minimize cost or application completion time.

We take the position that for network-intensive applications, the correct approach is for applications to concisely express some set of attributes, and for the cloud provider to take this information into account when returning a set of machines. Once assigned a set of machines, customers should be able to invoke a service to best distribute their application across these machines, such that some objective is met (e.g., minimizing cost). Such an architecture would improve performance, and may also give us a better way to balance the customer's needs with the provider's.

In this paper, we argue that the demands—especially the network demands—of an application should be expressed in such a way that placements can be found to satisfy a wide variety of objectives, as a single objective is likely not suitable to all customers. We outline several challenges that need to be met in order to design and deploy an architecture that supports this functionality. Among them, determining the correct granularity at which an application should express its demands, scalably measuring the network, and determining what information or APIs the cloud provider should export to customer applications. We also present some potential objectives that clients might want to optimize, as well as preliminary results using Amazon EC2 that indicate that performance can be substantially improved by taking application demands into account.

2. REQUIREMENTS

Applications run directly on physical machines in some cloud infrastructures and datacenters, while others use virtualization (hypervisors), or offer particular frameworks, such as Hadoop [10]. In this paper, we focus on the case where applications are run on virtual machines, without any frameworks

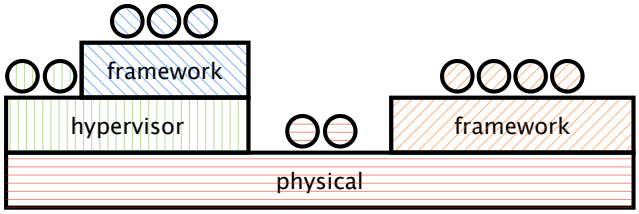


Figure 1: Applications (the balls in the figure) can be run through various combinations of hypervisors, frameworks (such as Hadoop), and physical machines.

involved (corresponding to the vertical-striped portions in Figure 1). This type of technology is akin to that offered by Amazon's EC2. Although we use this mental model throughout the paper, we expect that the architecture presented would work with the other application-deployment technologies shown in Figure 1.

2.1 Customer Objectives

Our proposed architecture should be able to satisfy a variety of customer objectives. For example:

- **Minimizing application completion time.** For applications that aren't continually-running services (e.g., web servers), customers are typically interested in the application finishing as quickly as possible. For example, it is common for MapReduce jobs to run for several hours. With network-aware application distribution, we may be able to significantly reduce completion time.
- **Minimizing monetary cost.** Cloud customers are typically charged for each machine launched, the amount of network bandwidth used, the amount of storage used, etc. These costs can vary considerably depending on the types of machines and storage used, as well as the time the application takes to run and how many machines are launched. Reducing the cost to run a set of applications may be a primary customer objective.
- **Maximizing the total number of application runs given a budget.** A project may have part of its budget dedicated towards analyzing some large dataset. In this case, the project would like to analyze as many pieces of data as possible without exceeding the budget.
- **Minimizing the variance of application completion time.** Many companies use public clouds for software testing or simulations; a typical scenario is running an extensive set of tasks every night. In this case, it is not so important that the tests finish as fast as possible, but rather that the time they will take to finish is *predictable*. This objective is also important when the application has its own customers who expect it to meet some particular objective.
- **Fault-tolerance.** Applications or services that need to quickly recover from network failures often require different components to be run on different parts of the network, or for portions of their application to be replicated in different datacenters.
- **Elasticity.** A compelling use of public cloud infrastructures is to allow customers to quickly scale applications (both "up" and "down"). Cloud providers typically offer ways to automatically scale applications based on the amount of CPU used per machine. Some applications may prefer to scale based on network latency, not just CPU demands.

In some cases, cloud users today try to achieve some of these goals by using low-level mechanisms. For instance, in order to "guarantee" that two virtual machines (VMs) have a high throughput path between them, a user might request that the VMs be placed on the same subnet. This is both too restrictive—the VMs might be able to obtain enough throughput between them even if they were on separate subnets—and possibly incorrect—being on the same subnet does not guarantee a fast path, particularly in the face of cross traffic.

2.2 Provider Requirements

Cloud providers have their own objectives to satisfy. For public cloud infrastructures, a primary goal is to maximize revenue. Though this goal is simple to state, there are many mechanisms put in place by cloud providers to achieve it. We list two below.

- **Migration.** In order to avoid hotspots, fix various network issues, etc., cloud providers with virtualized datacenters often want the ability to migrate customer applications. Being able to move customer applications allows for better consolidation, which can ease the cost of managing a datacenter. Any architecture for distributing customer applications should allow and adapt to migration.
- **Isolation.** Though cloud providers make no hard guarantees with respect to fairness, it is a reasonable goal for all customers in the cloud to experience a similar level of performance. Any architecture that prevents some users from achieving reasonable network performance might be prohibited or throttled by the cloud provider. Network-level isolation is a relatively open problem for cloud infrastructures today.

In §4.5, we discuss the extent to which the requirements of the customer and provider can both be satisfied in our architecture.

3. Proposed Architecture

Figure 2 depicts the proposed architecture for a system that satisfies customer objectives, including some "black boxes". Each of these black boxes brings up various research questions to solve, which we discuss in more detail in §4.

- **Expression.** Each application must be able to detail its demands and objectives in such a way that the system can find a satisfactory placement for the application. This expression should capture the network and CPU demands (and perhaps others) at an appropriate level of granularity such that they can be fed into the optimizer.
- **Measurement.** To determine a placement for a particular objective, the system needs to know the state of the net-

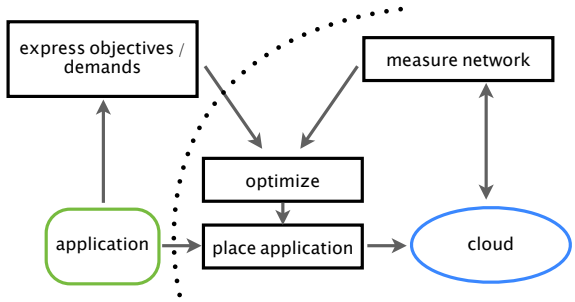


Figure 2: Each of the black boxes are part of our proposed architecture. Arrows show information flow between different parts of the architecture as well as the application and the cloud network. Portions of the architecture to the right of the dotted line could potentially be part of the cloud provider's infrastructure; see §4.

work; for instance, the throughputs along various paths in the network. The measurement module is responsible for monitoring the network and reporting the appropriate data to the optimization module.

- **Optimization.** Once the system knows the demands and goals of an application, as well as the current state of the network, it uses that information to determine a satisfying placement, i.e., the appropriate way to distribute the application across the machines such that the application's objective is met.
- **Placement.** Given a particular placement, the system must actually distribute the application's components (or "tasks", defined in the next section) in this manner and run it.

4. Research Agenda

In this section, we give some proposals for the four "black boxes" in Figure 2, and outline the set of questions that needs to be resolved for each. Our focus is on articulating the research challenges, rather than proposing specific solutions.

4.1 Expression

Many of the potential objectives in §2 can be expressed as functions of the given demands (see §4.3). We imagine that applications may express any or all of the following: network demands, as the number of bytes to be transferred or a desired sustained bandwidth; CPU demands, as percentage of CPU cores needed; fault-tolerance demands, as sets of machines that must exist in separate network locations and the desired separation (e.g., different datacenters); elasticity demands, as a function on which to scale machines (e.g., "launch a new machine if the average network latency is greater than 200ms"); and cost constraints, as the customer's budget in dollars.

At what granularity should applications specify their demands? Using the number of bytes transferred by an application as an example, one could express this demand in any of the following ways: as the total number of bytes transferred, as the number of bytes transferred between pairs of physical machines, or as the number of bytes transferred by every individual TCP connection or process. We believe that neither of these is a particularly good abstraction for expressing network demands. Instead, we propose expressing demands at the granularity of *tasks*, where tasks are groups of individual connections and processes, representing units of computation that cannot be split across machines. A task can be made up entirely of local computations, or it can transfer (receive) data to (from) other tasks. Examples of tasks might include an instance of a reduce task in MapReduce running on one machine, or the computation associated with answering one query (or set of queries) in a distributed database. Using tasks to express demands is different from systems such as Conductor [18], which coordinate cloud applications using more coarse-grained specifications (e.g., the total amount of data transferred by an application).

It is possible that an application may not be able to accurately express its demands at this level of granularity. In this case, either the application can express demands with less accuracy, e.g., that two tasks need to transfer a lot of data vs. a little, rather than specifying the exact number of bytes, or the placement system could "sample" the application once (i.e., run some portion of it) and measure the demands itself. We do, however, believe that the application's author could correctly define the tasks of an application, perhaps via an API exported by the cloud provider (see §4.5).

4.2 Measurement

The placement and optimization modules require measurement data about the current state of the network. An obvious question is how much data is needed, and how often does the data need to be updated? Many of the objectives mentioned in §2 require knowledge of the achievable rates along network paths. But there is other data available from the network; for instance, the network topology. How much effort does it take to measure this type of information, and to what extent does this extra information help? Related work [4] suggests that to satisfy certain objectives—in particular, to improve predictability—knowledge of the network topology may be necessary.

The question of how much information this system will require is closely related to the question of who should be doing the measurement. In today's public clouds, any detailed information about the network has to be inferred by the customer's application (or perhaps by an additional service available to the customer). At the other end of the spectrum, one can imagine the cloud provider doing all of the measurement, and simply taking customer applications and placing them such that their objectives were satisfied (customers would have to trust that their applications were being placed correctly).

A third option is for cloud providers to export an API to allow clients to get some measurement information directly, i.e., without having to actively measure themselves. Note that there may be some information about the network that *only* cloud providers have access to—for instance, the physical topology, and information about the cross traffic of other

users—and cloud providers may not want to allow customers access to all of this information.

Regardless of who performs the measurements, we must address scalability. Measuring network paths often involves sustained TCP transfers. But cloud networks are large; with sustained transfers, the network can change substantially between the time when the first and last paths are measured. With too long a delay, we cannot be confident that the placements our system finds are still optimal. Moreover, measurement traffic will interfere with traffic from real users, which is likely unacceptable to most providers. How can we measure cloud networks quickly and accurately? We explore some of these concerns in §5.4.

4.3 Optimization

Once our system has an expression of an application's objective(s) and demands, how can it determine a satisfying assignment? Many, if not all, of the objectives listed in §2 can be formulated as convex programs. After formulating the appropriate problem, the system can simply use off-the-shelf optimization software such as CPLEX [8] to solve it. To illustrate this idea, we give a few examples of optimization problems for various objectives. In all examples, we are trying to place J tasks on M machines.

Minimizing completion time. Here, the CPU demands are expressed as a matrix $CR_{J\times 1}$, where CR_j is the CPU demand for task j. Network demands are expressed as $B_{J\times J}$, where B_{ij} is the amount of data task i needs to transfer to task j. The available CPU on each machine is expressed as $C_{M\times 1}$, and the bandwidth on each path is expressed as $R_{M\times M}$. Given these variables, a placement of the J tasks onto the M machines is an indicator matrix X ($X_{jm} = 1$ if task j is placed on machine m), such that:

$$\sum_{m=1}^{M} X_{jm} = 1, \forall j \in [1, J] \quad (1)$$

That is, each task must be placed on exactly one machine. In addition, the placement of tasks on each machine must obey CPU constraints, i.e.,

$$\sum_{j=1}^{J} CR_j \cdot X_{jm} \leq C_m, \forall m \in [1, M] \quad (2)$$

To minimize completion time is equivalent to solving

$$\min \ \max_{m,n} D_{mn}/R_{mn},$$

where $D = X^T BX$, and D_{mn} gives the amount of data to be transferred between *machines* m and n in a particular placement, assuming that the completion time is bottlenecked by the network.

We have implemented a system that solves this particular optimization problem on public cloud infrastructures. On Amazon's EC2 network, we observed an average 21% - 31% reduction in application completion times compared to other placement strategies, which we show in §5.

Fault-tolerance. A customer can express their fault-tolerance constraints by stating the tasks that must be located on separate physical machines, across datacenters, or multiple hops away. As a simple example, to prevent two tasks i and j from being placed on the same machine m, we use the following constraint:

$$X_{im} + X_{jm} \leq 1 \ \forall m \in [1, M]$$

This constraint can easily be extended for multiple pairs of tasks that cannot be placed on the same machine.

Minimizing Cost. The cost to run machines in a cloud network comes from launching VMs, keeping the VMs up for a certain time period, and transferring data between VMs. Suppose launching VMs costs C_l dollars per VM, running VMs costs C_c dollars per second per CPU, and network transfers costs C_t dollars per byte. The total number of VMs launched is $n = \sum_{m=1}^{M} (\prod_{j=1}^{J} X_{jm})$. The total completion time for a particular placement is $t = \max_{m,n}(D_{mn}/R_{mn})$. The total amount of data transmitted between VMs is:

$$d = \sum_{i=1,j=1}^{J,J} \sum_{m=1}^{M} B_{ij} X_{im}(1 - X_{jm})$$

Here, $X_{im}(1 - X_{jm})$ ensures that we only consider the data transmitted between VMs. If two tasks are placed on the same VM, then there is only data exchange via disk I/O. The problem now is to solve:

$$\min(n \cdot C_l + t \cdot C_c + d \cdot C_t)$$

while satisfying (1) and (2).

4.4 Placement

Once a placement has been found, the application's tasks need to be distributed across the recommended machines. To do this placement, cloud providers could export an API that specifies a `Task` class, which customers can incorporate into their applications. Like a traditional `Thread` class, individual `Tasks` could be run independently—i.e., on different machines—but the units of computation comprising a single task could not.

In addition, cloud providers may need to support tagging network connections belonging to the same application, regardless of the specific machines they are running between. For better network isolation and to make application completion times more predictable, these tags, which allow an application to specify a "bundle" of connections, could be used for better network-level scheduling.

4.5 Understanding Interactions

The cloud provider's goal of maximizing revenue seems at odds with allowing customers to optimize their network placements. After all, these placements may frequently lead to customers spending less time using the cloud network, which, at first glance, results in a loss of money for the provider. However, deploying such a system would likely result in more satisfied customers, better load balancing on the network, and possibly better performance for customers who *weren't* using the system. All of these results could increase revenue for the cloud provider.

There is also the question of how the optimizations of individual customers will affect the cloud provider's network as a whole, as well as the provider's objective of maximizing revenue. Is it possible that individual customer optimization could detrimentally affect the network on a global scale? If the cloud provider were in charge of finding placements for each individual customer, they could optimize multiple

customers together; how much improvement would we get from such a strategy?

5. PRELIMINARY RESULTS

To test the potential of our proposed architecture, we ran some preliminary experiments on EC2. In these experiments, we measured the path throughputs on an EC2 topology,[1] and determined the optimal placement for the minimizing completion time of a particular workload. We then compared the time it took for this placement to complete against the time it would take for a variety of other placements to complete.

5.1 Workloads

For our experiments, we would like to be able to test on some common distributed-application workloads, and also to abstract the important properties of a workload out, so that we can run these abstracted workloads under multiple configurations (different topologies, varying numbers of tasks, etc.). There is little published information on application-level traffic patterns in cloud computing applications, but we were able to classify some workload abstractions based on prior work.

- **Arbitrary Pairs**. This workload simulates arbitrary pairs of tasks transferring data, similar to how scientific computing applications and database joins work. There, data is transferred between a fraction f of pairs of tasks. This abstraction is similar to the elephant-and-mice workloads that appear in many datacenter papers [2, 3, 9].
- **One-to-many**. This workload represents a backup of a filesystem, where each node transfers one copy of its data to some number of distinct servers. In our experiments, each client makes three copies, as in [5], a pattern which is similar to other network-intensive cloud applications [7, 14].
- **All-to-all**. Here, data is transferred between every pair of tasks in the network, similar to an all-to-all shuffle in Hadoop [2, 9, 16]. Generally, this workload is referenced by shuffling machine memory to machine memory (rather than from one task to another task), however it actually occurs at the application layer; see [16]. This workload is equivalent to an arbitrary pairs workload where $f = 1$.
- **All-to-all Multi-user**. Again, data is transferred between every pair of tasks in the network. The difference between this workload and the all-to-all workload is that in this workload, we simulate multiple users running all-to-all transfers. Generally, we imagine there being more machines than users (e.g., ten machines with two users, each user with five tasks).

Note that, except in the all-to-all workload, it is *not* the case that all pairs of tasks in the workload will transfer data to one another. We will refer to the pairs of tasks that do transfer data as *transferring pairs*.

[1]By "EC2 topology" we mean a collection of Amazon EC2 instances (typically ten instances in our experiments) under the control of the same user.

There are also some additional parameters defined in §4.3 shared by all of the workloads. The variable B describes the amount of data sent between transferring pairs. We modeled B both in the uniform case (all transferring pairs transfer the same amount data) and as a Zipf distribution, where all pairs transfer an amount of data drawn independently from a Zipf distribution with $\alpha = 1$. The variable C represents the CPU demand of a task, given by the percentage of the CPU used. In any particular experiment, we set C to be equal for all tasks, but we explore the effects of different values of C.

5.2 Alternate Placement Strategies

To understand the improvement that clients would get from an optimal placement, we compared the time it took for the optimal placement to complete against three alternate placements, none of which is network-aware. Whenever we speak of a placement, we mean a valid placement (one that satisfies CPU constraints), but not an necessarily optimal one.

- **Random**. Here, tasks are placed randomly on machines.
- **Min-Max CPU**. This placement simulates CPU load-balancing, where a roughly equal percentage of the CPU is used on all machines. If the CPU requirements of each task are equivalent, this placement will effectively place tasks on machines in a round-robin fashion.
- **Min Num Machines**. This placement simulates minimizing the number of physical machines used. For example, a set of eight tasks that each need 25% of the CPU can be placed entirely on two physical machines. Minimizing the number of physical machines used is desirable for clients who want to save money as well as cloud providers who want to fully utilize each machine [1].

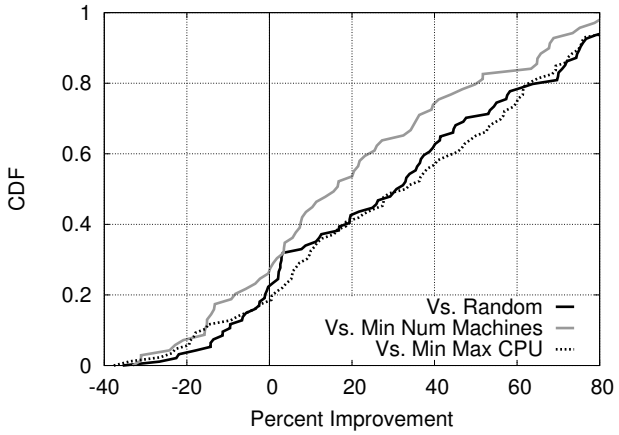
In each experiment, we must define two more parameters: M and J. M is the number of machines in the topology; unless specified, we use $M = 10$. J is the number of tasks in the workload. We use values of both $J = 5$ and $J = 10$. Values of J much higher than M are somewhat unreasonable in network-intensive applications. If $J >> M$, we have many tasks transferring data while using very little CPU, which indicates that tasks are transferring data but not processing most of it.

We tested these placement methods on a variety of Amazon EC2 topologies. In our experiments, all transferring tasks send data to one another. Consequentially, our results reflect the improvement our system can gain in the face of actual path rate variations, unknown topological bottlenecks, etc.

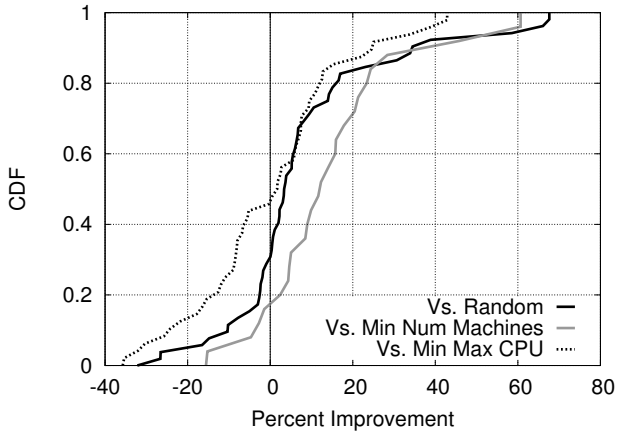
5.3 Results

For a particular configuration (defined by the workload type, B, J, etc.), we first measure an EC2 topology (we used packet trains for these measurements; see §5.4). We then calculate the optimal placement and run the tasks in that placement; this run gives us a workload completion time t_{opt}. We then take the alternate placement and run the tasks in that placement; this run gives us a time t_{alt}. The measured performance improvement is $1 - t_{opt}/t_{alt}$.

Figure 3 shows two CDFs. The first (Figure 3(a)) shows



(a) Performance improvements on arbitrary pairs workloads.



(b) Performance improvements across structured workloads.

Figure 3: Performance improvements across all workloads. Figure 3(a) displays the improvement on arbitrary-pairs workloads, for varying f's (divided equally between .3 and .7), as well as all-to-all workloads where traffic was Zipf-distributed with $\alpha = 1$. Figure 3(b) displays improvement on the more structured workloads: both types of all-to-all with tasks transferring equal amounts of data, and one-to-many.

the percent improvement across all instances of arbitrary pairs workloads. The second (Figure 3(b)) shows the performance improvement across the more structured workloads (one-to-many, all-to-all, all-to-all multi-user). We ran 514 experiments, each on a different EC2 topology.

5.3.1 Performance on Arbitrary Pairs Workloads

Figure 3(a) shows a significant performance improvement of our system over all other methods. The mean (median) performance improvement is 29.3% (30.4%) over random placement, 31.1% (33.5%) over min-max CPU, and 19.4% (15.4%) over the min-num machines. Where does this performance improvement come from? One feature that our system takes advantage of is packing pairs of transferring tasks on the same machine, thus effectively using intra-machine links (this phenomenon can only occur in experiments where $C < 100\%$). We saw improvement in almost all instances when the number of network transfers that the optimal placement used was fewer than the number that the alternate placement used.

5.3.2 Structured Workloads

We are also interested in what happens when many network paths have to be used. This case typically corresponds to the highly-structured workloads, such as all-to-all, all-to-all multi-user, and one-to-many. In these structured workloads, because many tasks are transferring data (in some cases, all tasks), many paths in the network need to be used; when there is one task per machine, and as many machines as tasks, all paths must be used. When the majority of paths in the network have to be used, there is little room for improvement for *any* placement scheme.

We can see in Figure 3(b) that the min-max CPU algorithm does as well as the optimal placement scheme; each of these placement methods do a good job of spreading load around different paths. For any network-aware scheme, putting multiple connections on one path generally increases the completion time more than placing the connections on separate paths, even if one of the paths is somewhat slow. The min-max CPU method, while not taking the network into account, does a reasonable job of balancing network load by virtue of distributing tasks across machines.

In contrast, the other two placement schemes—random and min-num machines—do a relatively poor job in these workloads. Note that min-num machines is designed to *not* spread load across the network. Networks with more skew in the path-rate distribution will generally see more room for improvement in these types of workloads.

5.4 Scalability

Though these results are preliminary, we expect that they will scale to clients running more VMs on larger networks. In these cases, it is possible that the optimization problems in §4.3 will occasionally be difficult to solve efficiently. To address this concern, we have experimented with a greedy network-aware strategy, which attempts to use the fastest paths first, but takes into consideration the effect of placing multiple transfers on the same path. Though this strategy is not always optimal, in many cases it performs comparably to the optimal algorithm. More research should be done to determine when the greedy algorithm is appropriate.

Additionally, we also explored how to measure cloud networks more efficiently. We chose to use the well-known method of packet trains [12], finding that they were relatively accurate in cloud networks (roughly 6% error). Further results on packet trains in cloud networks are out of the scope of this paper.

6. Related Work

The idea that one should allow application expression in cloud networks has not received much attention. Many systems aim to improve performance in datacenter networks, but without taking input from the customers. FairCloud [15] proposes a mechanism for handling the tradeoff between providing bandwidth guarantees and sharing the network. Oktopus [4] allows for more predictable application run times in certain topologies. VL2 [9] and Hedera [2] present different flow scheduling systems for achieving scalability and maximizing utilization. There have also been proposals to use software-defined networking [11] to achieve such gains.

Orchestra [6] proposed a scheduling mechanism for Hadoop, aiming to improve the transfer times of communication patterns such as broadcast and shuffle. Their design tries to find the best time to start a transfer given that the job placements are fixed, while our system tries to find the best placement of jobs.

A few systems allow customer expression in some form. Conductor [18] allows customers to specify coarse-grained facts about their applications, and uses this information to choose particular cloud resources (e.g., local storage vs. offsite storage). The architecture that we have proposed is complementary to Conductor; it allows clients to describe applications with a much finer granularity, and deals with optimally placing an application once a service has been chosen. Moreover, it works with a variety of datacenter technologies; currently Conductor is limited to MapReduce frameworks. Mohammadi, et al. [13] allow applications to express their network requirements, but it is not clear how well this system will operate in a complex topology such as a datacenter network. Webb, et al. [17] allow topology switching in datacenter networks, with the goal of using the best available topology for a particular application, but do not allow for a fine-grained specification of application demands.

In contrast to these works, our proposed architecture allows for a much more descriptive specification from cloud customers and multiple objectives to be met in a variety of frameworks. We do not force a particular framework on the client, nor require changing routing schemes, nor knowing the topology; it is likely that control over routes and/or topology could *benefit* our system.

7. Conclusion

This paper discussed the need for cloud users to be able to express the demands and objectives of their applications to the infrastructure for better performance. We outlined some key research questions that need to be addressed to implement such a system, and gave preliminary results indicating that these ideas can improve application performance. We believe that such a system is feasible, and would benefit both customers and cloud providers.

8. References

[1] Aboulnaga, A., Salem, K., Soror, A. A., Minhas, U. F., Kokosielis, P., and Kamath, S. Deploying Database Appliances in the Cloud. In *IEEE Data Engineering Bulletin* (2009).

[2] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI* (2010).

[3] Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. Data Center TCP (DCTCP). In *SIGCOMM* (2010).

[4] Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. Towards Predictable Datacenter Networks. In *SIGCOMM* (2011).

[5] Blanas, S., Patel, J. M., Ercegovac, V., Rao, J., Shekita, E. J., and Tian, Y. A Comparison of Join Algorithms for Log Processing in MapReduce. In *SIGMOD* (2010).

[6] Chowdhury, M., Zaharia, M., Ma, J., Jordan, M. I., and Stoica, I. Managing Data Transfers in Computer Clusters with Orchestra. In *SIGCOMM* (2011).

[7] Cisco Data Center Infrastructure 2.5 Design Guide. `http://www.cisco.com/univercd/cc/td/doc/solution/dcidg21.pdf`.

[8] IBM ILOG CPLEX Optimizer. `http://cplex.com`.

[9] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. VL2: A Scalable and Flexible Data Center Network. In *SIGCOMM* (2009).

[10] Apache Hadoop. `http://hadoop.apache.org`.

[11] Handigol, N., Flagslik, M., Seetharaman, S., Johari, R., and McKeown, N. Aster*x: Load-Balancing as a Network Primitive. In *ACLD (Poster)* (2010).

[12] Jain, R., and Routhier, S. A. Packet Trains: Measurements and a New Model for Computer Network Traffic. *IEEE Journal on Selected Areas in Communications 4* (1986), 986–995.

[13] Mohammadi, E., Karimi, M., and Heikalabad, S. R. A Novel Virtual Machine Placement in Cloud Computing. *Australian Journal of Basic and Applied Sciences* (2011).

[14] Moretti, C., Bulosan, J., Thain, D., and Flynn, P. All-pairs: An Abstraction for Data-intensive Cloud Computing. In *IPDPS* (2008).

[15] Popa, L., Krishnamurthy, A., Ratnasamy, S., and Stoica, I. FairCloud: Sharing the Network in Cloud Computing. In *HotNets* (2011).

[16] Rasmussen, A., Porter, G., Conley, M., Madhyastha, H. V., Mysore, R. N., Pucher, A., and Vahdat, A. TritonSort: A Balanced Large-Scale Sorting System. In *NSDI* (2011).

[17] Webb, K. C., Snoeren, A. C., and Yocum, K. Topology Switching for Data Center Networks. In *HotIce* (2011).

[18] Wieder, A., Bhatotia, P., Post, A., and Rodrigues, R. Conductor: Orchestrating the Clouds. In *LADIS* (2010).